T-Head ISA extension specification (Xthead*)

Version 2.2.1, 2022-12-09

Table of Contents

Chapter 1. Document information

1.1. Copyright and license information

1.2. Contributors

The list below includes all contributors to this document in alphabetical order:

- Christoph Müllner <christoph.muellner@vrull.eu>
- Cooper Qu <cooper.qu@linux.alibaba.com>
- Lifang Xia <lifang_xia@linux.alibaba.com>
- Philipp Tomsich <philipp.tomsich@vrull.eu>
- Yunhai Shang < yunhai@linux.alibaba.com>
- Zhiwei Liu <zhiwei_liu@linux.alibaba.com>

1.3. Changelog

- 2022-11-04: Fixes for xtheadint, xtheadfmv
- 2022-09-23: Adding xtheadint extension
- 2022-09-23: Adding xtheadfmv extension
- 2022-09-03: xtheadmemidx: Fix load width of lurwu, xtheadmac: Fix and simplify extension behaviour
- 2022-08-25: Adjusting contributors list
- 2022-08-10: *.adoc: Fix several formatting issues
- 2022-08-01: Remove XTheadEE and rework availability/permission documentation, adding lifecycle-states, adding doc versioning and changelog
- 2022-07-29: Adding XTheadEE (mxstatus.theadisaee) and mxstatus.ucme (as part of XTheadCmo)
- 2022-07-27: Adding XTheadMemPair, XTheadMemIdx, XTheadMac, XThadCondMov, XTheadB[a,b,s]
- 2022-07-26: Adding XTheadCmo U mode permission, Adding XTHeadSync, XTheadFMemIdx
- 2022-07-20: Adding XTheadMemPair extension, adding HW requirements for XTheadCmo
- 2022-07-16: Adding complete XTheadCmo extension
- 2022-07-12: Initial draft with four CMO instructions

Chapter 2. Introduction

The T-Head extension collection was created to augment the RISC-V ISA by adding additional functionality to enable faster and more energy-efficient solutions.

The RISC-V ISA and its standardized extensions provide a rich set of instructions suitable for a wide range of applications, starting from specialized microcontrollers to HPC systems. The suitability for such a range of systems comes from the fact that the RISC-V ISA is modularized with a base instruction set and a range of extensions that target specific target applications and functionality.

The RISC-V ISA and its authors strongly advertise the ability to create vendor extensions. Dedicated encoding spaces ensure, that there are not conflicts with standard extensions.

This document specifies the T-Head extension collection, a collection of vendor extensions that are implemented in many T-Head processors.

2.1. Overview

The T-Head extension collection follows the principles of the RISC-V ISA. The collection consists of the following ISA extensions:

- `XTheadCmo` provides instructions for cache management.
- `XTheadSync` provides instructions for multi-processor synchronization.
- `XTheadBa` provides instructions for address calculations.
- `XTheadBb` provides instructions for basic bit-manipulation.
- `XTheadBs` provides single-bit instructions.
- `XTheadCondMov` provides instructions for conditional moves.
- `XTheadMemIdx` provides GPR memory operations.
- `XTheadMemPair` provides two-GP-register memory operations.
- `XTheadFMemIdx` provides floating-point memory operations.
- `XTheadMac` provides multiply-accumulate instructions.
- `XTheadVdot` provides instructions for vector dot.

2.2. Dependencies to standard extensions

The T-Head extension collection is designed to be compatible with RISC-V's base integer instruction sets RV32I and RV64I.

Some instructions are only available if the system's `XLEN` is 64 (i.e. integer registers and supported user address space is 64 bits). To highlight the availability, each extension documents this for each instruction.

Instruction that operate on floating-point registers can be used in combination with `F` (32-bit floating-point register) or `D` (64-bit floating-point registers). Some instruction are only available if

the system implements the `D` extension. To highlight the availability, each floating-point extension document this for each instruction.

2.3. Enablement of extensions and instructions

All extensions and instructions are expected to be enabled at all times. The instructions can be used at any time when executing in the documented privilege levels, that permit the execution of the instruction.

However, there might be SoC-specific mechanisms to ensure this behaviour. E.g. the T-Head C906 requires the CSR field `mxstatus.theadisaee` to be set to `1` to enable the ISA extensions. Another example is the CSR field `mxstatus.ucme` of the T-Head C906, that is required to be set to `1` in order to execute cache management instructions in `U` mode (if the instruction is documented that this is permitted).

2.4. Lifecycle

The lifecycle of the extensions included in this document is defined on a per-extension base.

Possible states are:

- `Draft`: No guarantees (everything may change).
- `Stable`: No feature additions/removals/changes. Only clarifications.

A `draft` specification is expected to become `stable` in the future (with exepcted changes until then). A `stable` extension remains in that state for ever.

Chapter 3. Cache Management Operations (XTheadCmo)

The `XTheadCmo` extension is `stable`.

The `XTheadCmo` ISA extension provides cache management operations.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction	HW requirements
Y	Y	th.dcache.call	Clean all D-cache	D-cache
Y	Y	th.dcache.ciall	Clean & invalidate all D-cache	D-cache
Y	Y	th.dcache.iall	Clean all D-cache	D-cache
Y	Y	th.dcache.cpa *rs1*	Clean D-cache at PA	D-cache
Y	Y	th.dcache.cipa *rs1*	Clean & invalidate D-cache at PA	D-cache
Y	Y	th.dcache.ipa *rs1*	Invalidate D-cache at PA	D-cache
Y	Y	th.dcache.cva *rs1*	Clean D-cache at VA	D-cache, MMU
Y	Y	th.dcache.civa *rs1*	Clean & invalidate D-cache at VA	D-cache, MMU
Y	Y	th.dcache.iva *rs1*	Invalidate D-cache at VA	D-cache, MMU
Y	Y	th.dcache.csw *rs1*	Clean D-cache by set/way	D-cache
Y	Y	th.dcache.cisw *rs1*	Clean & invalidate D-cache by set/way	D-cache
Y	Y	th.dcache.isw *rs1*	Invalidate D-cache by set/way	D-cache
Y	Y	th.dcache.cpal1 *rs1*	Clean L1 D-cache at PA	D-cache, 2nd level cache
Y	Y	th.dcache.cval1 *rs1*	Clean L1 D-cache at VA	D-cache, 2nd level cache, MMU
Y	Y	th.icache.iall	Invalidate all I-cache	I-cache
Y	Y	th.icache.ialls	Invalidate all I-cache on all harts	I-cache, multicore
Y	Y	th.icache.ipa *rs1*	Invalidate I-cache at PA	I-cache
Y	Y	th.icache.iva *rs1*	Invalidate I-cache at VA	I-cache, MMU
Y	Y	th.l2cache.call	Clean all L2 cache	D/I-cache, 2nd level cache
Y	Y	th.l2cache.ciall	Clean & invalidate all L2 cache	D/I-cache, 2nd level cache
Y	Y	th.l2cache.iall	Invalidate all L2 cache	D/I-cache, 2nd level cache

The last column of the table above names the HW requirements of the instructions. E.g. to clean the data cache using `dcache.call`, a D-cache is required. Instructions that are executed without the

required HW requirements available (e.g. `l2cache.call` on a system without a L2 cache) do not chance any architecturally visible state, except for advancing the program counter and incrementing any applicable performance counters (i.e. it behaves like executing a `NOP` instruction).

3.1. Instructions

3.1.1. th.dcache.call

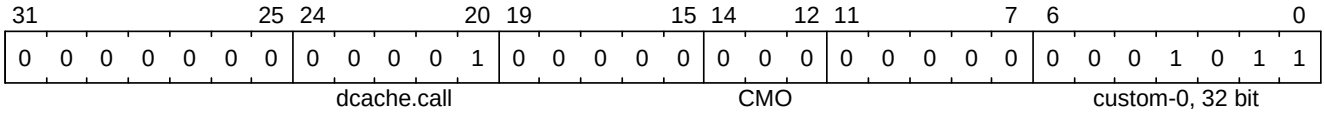
Synopsis

Clean all D-cache

Mnemonic

th.dcache.call

Encoding



Description

This instruction cleans all cache lines of the data cache on the local hart.

Dirty cache lines will be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<write back all dirty data cache lines of the local hart>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache

3.1.2. th.dcache.ciall

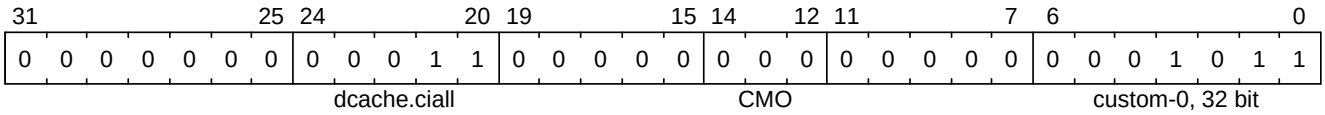
Synopsis

Clean & invalidate all D-cache

Mnemonic

th.dcache.ciall

Encoding



Description

This instruction cleans and invalidates all cache lines of the data cache on the local hart. Dirty cache lines will be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<write back all dirty data cache lines of the local hart>
<invalidate all data cache lines of the local hart>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache

3.1.3. th.dcache.iall

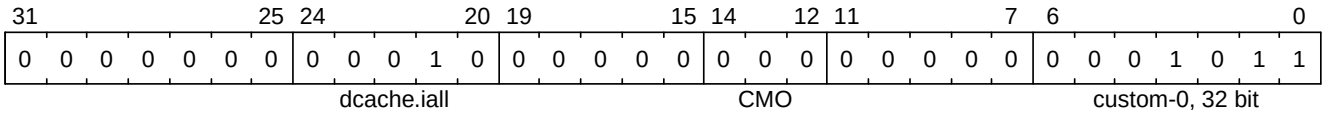
Synopsis

Invalidate all D-cache

Mnemonic

th.dcache.iall

Encoding



Description

This instruction invalidates all cache lines of the data cache on the local hart. Dirty cache lines will not be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<invalidate all data cache lines of the local hart>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache

3.1.4. th.dcache.cpa

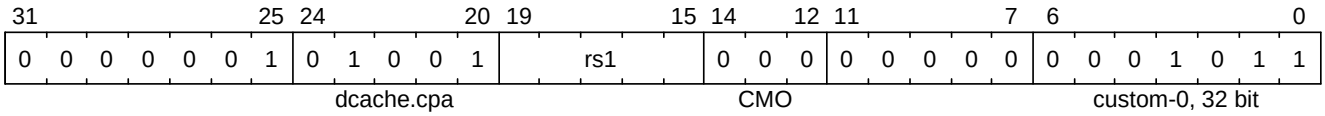
Synopsis

Clean D-cache at PA

Mnemonic

th.dcache.cpa *rs1*

Encoding



Description

This instruction cleans the cache lines that match the specified physical address in the D-cache. If a cache line is dirty it will be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<write back all dirty data cache lines matching the PA>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache

3.1.5. th.dcache.cipa

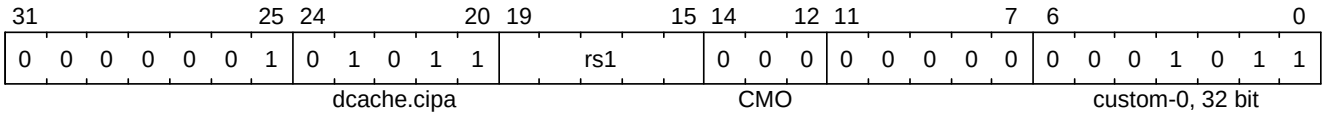
Synopsis

Clean and invalidate D-cache at PA

Mnemonic

th.dcache.cipa *rs1*

Encoding



Description

This instruction cleans and invalidates the cache lines that match the specified physical address in the D-cache. If a cache line is dirty it will be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<write back all dirty data cache lines matching the PA>
<invalidate all data cache lines matching the PA>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache

3.1.6. th.dcache.ipa

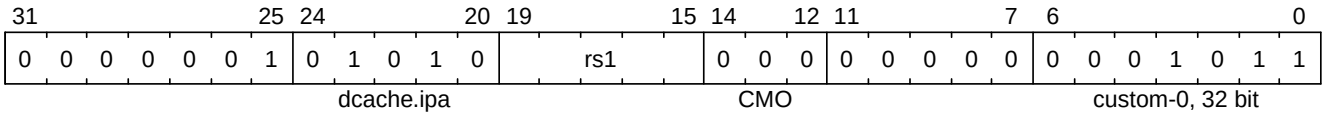
Synopsis

Invalidate D-cache at PA

Mnemonic

th.dcache.ipa *rs1*

Encoding



Description

This instruction invalidates the cache lines that match the specified physical address in the D-cache. Dirty cache lines will not be written back to the next-level storage.

Operation

```
if (priv_level == U)
  <raise illegal instruction exception>
}

<invalidate all data cache lines matching the PA>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache

3.1.7. th.dcache.cva

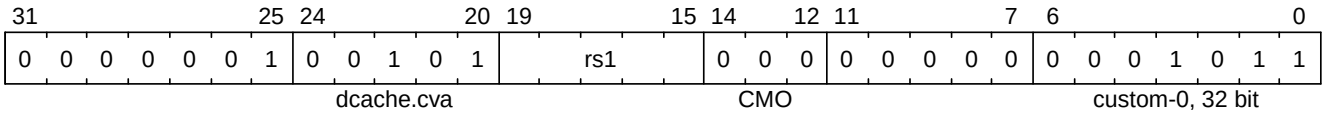
Synopsis

Clean D-cache at VA

Mnemonic

th.dcache.cva *rs1*

Encoding



Description

This instruction cleans the cache lines that match the specified virtual address in the D-cache. If a cache line is dirty it will be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<write back all dirty data cache lines matching the VA>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache, MMU

3.1.8. th.dcache.civa

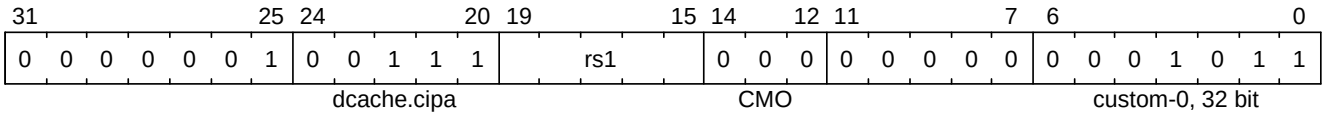
Synopsis

Clean and invalidate D-cache at VA

Mnemonic

th.dcache.civa *rs1*

Encoding



Description

This instruction cleans and invalidates the cache lines that match the specified virtual address in the D-cache. If a cache line is dirty it will be written back to the next-level storage.

Operation

```
<write back all dirty data cache lines matching the VA>
<invalidate all data cache lines matching the VA>
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache, MMU

3.1.9. th.dcache.iva

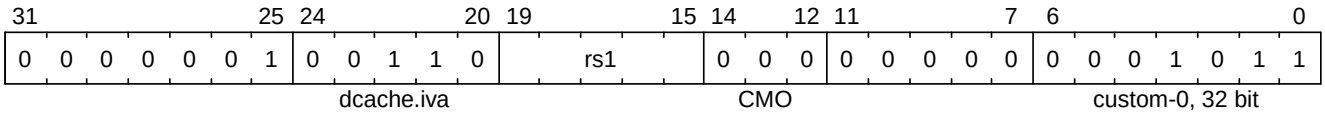
Synopsis

Invalidate D-cache at VA

Mnemonic

th.dcache.iva *rs1*

Encoding



Description

This instruction invalidates the cache lines that match the specified virtual address in the D-cache. Dirty cache lines will not be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<invalidate all data cache lines matching the VA>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache, MMU

3.1.10. th.dcache.csw

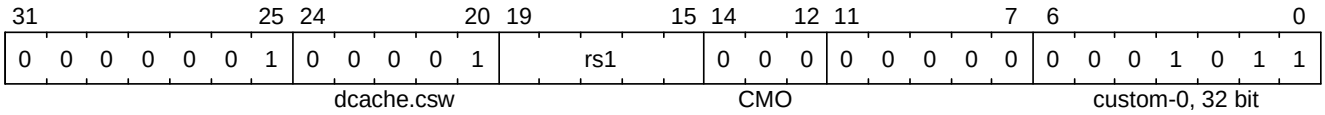
Synopsis

Clean D-cache by set/way

Mnemonic

th.dcache.csw *rs1*

Encoding



The register content of `rs1` defines the set/way and has the following bit assignment:

- rs1[64:31]…reserved (write `0`)
- rs1[31:32-w]…number of the way to operate on
- rs1[w-1:l+s]…reserved (write `0`)
- rs1[l+s-1:l]…number of the set to operate on
- rs1[l-1:4]…reserved (write `0`)
- rs1[3:1]…cache level to operate on (`0` for L1 cache, `1` for L2 cache, etc.)
- rs1[0]…reserved (write `0`)

The bit positions to specify the set and the way to operate on depend on the actual cache implementation. There are three cache properties that need to be considered:

- `nways` (number of ways)
- `nsets` (number of sets)
- `linesize` (size of a cache line in bytes)

Derived from these numbers the following constants are defined:

- `w := log2(nways)`
- `s := log2(nsets)`
- `l := log2(linesize)`

E.g. a 64 KiB, 2-way set-associative cache (`w = 1`) with a cacheline size of 64 bytes (`l = 6`) has 512 sets (`s = 9`) and needs to use the following bits to set the set and the way to operate on:

- [31]…number of the way (`0` or `1`)
- [14:6] number of the set (`0`..`512`)

Description

This instruction cleans the cache line that matches the specified set/way in the D-cache. If the cache line is dirty it will be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<write back data cache line matching the set/way if dirty>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache

3.1.11. th.dcache.cisw

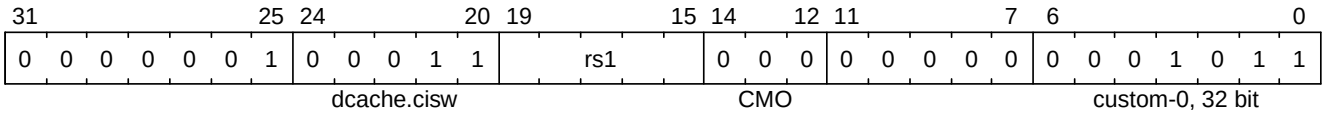
Synopsis

Clean & invalidate D-cache by set/way

Mnemonic

th.dcache.cisw *rs1*

Encoding



The register content of `rs1` defines the set/way and has the following bit assignment:

- rs1[64:31]…reserved (write `0`)
- rs1[31:32-w]…number of the way to operate on
- rs1[w-1:l+s]…reserved (write `0`)
- rs1[l+s-1:l]…number of the set to operate on
- rs1[l-1:4]…reserved (write `0`)
- rs1[3:1]…cache level to operate on (`0` for L1 cache, `1` for L2 cache, etc.)
- rs1[0]…reserved (write `0`)

The bit positions to specify the set and the way to operate on depend on the actual cache implementation. There are three cache properties that need to be considered:

- `nways` (number of ways)
- `nsets` (number of sets)
- `linesize` (size of a cache line in bytes)

Derived from these numbers the following constants are defined:

- `w := log2(nways)`
- `s := log2(nsets)`
- `l := log2(linesize)`

E.g. a 64 KiB, 2-way set-associative cache (`w = 1`) with a cacheline size of 64 bytes (`l = 6`) has 512 sets (`s = 9`) and needs to use the following bits to set the set and the way to operate on:

- [31]…number of the way (`0` or `1`)
- [14:6] number of the set (`0`..`512`)

Description

This instruction cleans and invalidates the cache line that matches the specified set/way in the D-cache. If the cache line is dirty it will be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<write back data cache line matching the set/way if dirty>
<invalidate data cache line matching the set/way>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache

3.1.12. th.dcache.isw

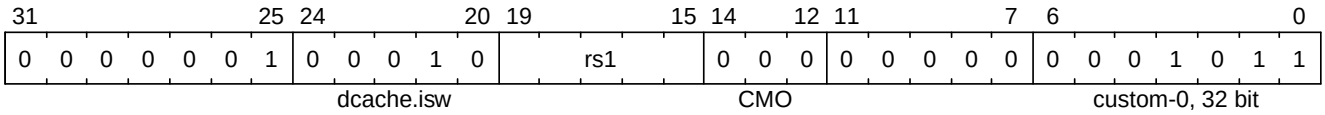
Synopsis

Invalidate D-cache by set/way

Mnemonic

th.dcache.isw *rs1*

Encoding



The register content of `rs1` defines the set/way and has the following bit assignment:

- rs1[64:31]…reserved (write `0`)
- rs1[31:32-w]…number of the way to operate on
- rs1[w-1:l+s]…reserved (write `0`)
- rs1[l+s-1:l]…number of the set to operate on
- rs1[l-1:4]…reserved (write `0`)
- rs1[3:1]…cache level to operate on (`0` for L1 cache, `1` for L2 cache, etc.)
- rs1[0]…reserved (write `0`)

The bit positions to specify the set and the way to operate on depend on the actual cache implementation. There are three cache properties that need to be considered:

- `nways` (number of ways)
- `nsets` (number of sets)
- `linesize` (size of a cache line in bytes)

Derived from these numbers the following constants are defined:

- `w := log2(nways)`
- `s := log2(nsets)`
- `l := log2(linesize)`

E.g. a 64 KiB, 2-way set-associative cache (`w = 1`) with a cacheline size of 64 bytes (`l = 6`) has 512 sets (`s = 9`) and needs to use the following bits to set the set and the way to operate on:

- [31]…number of the way (`0` or `1`)
- [14:6] number of the set (`0`..`512`)

Description

This instruction invalidates the cache line that matches the specified set/way in the D-cache. Dirty cache lines will not be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<invalidate data cache line matching the set/way>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache

3.1.13. th.dcache.cpal1

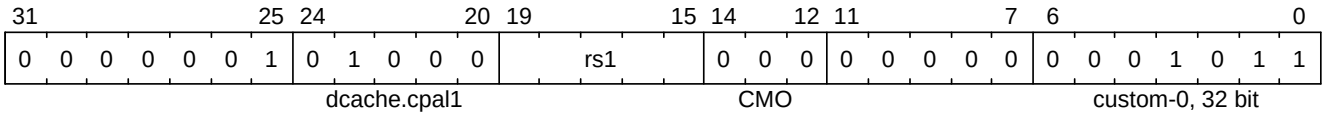
Synopsis

Clean L1 D-cache at PA

Mnemonic

th.dcache.cpal1 *rs1*

Encoding



Description

This instruction cleans the cache lines that match the specified physical address in the L1 D-cache. If a cache line is dirty it will be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<write back all dirty L1 data cache lines matching the PA>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache, 2nd level cache

3.1.14. th.dcache.cval1

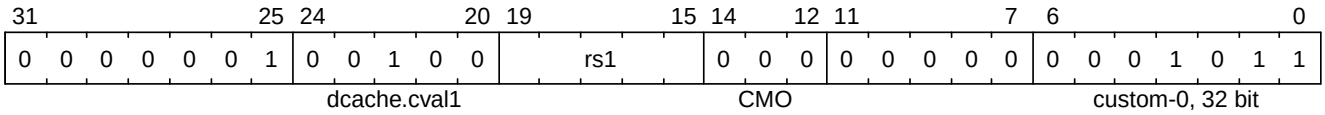
Synopsis

Clean L1 D-cache at VA

Mnemonic

th.dcache.cval1 *rs1*

Encoding



Description

This instruction cleans the cache lines that match the specified virtual address in the L1 D-cache. If a cache line is dirty it will be written back to the next-level storage.

Operation

```
<write back all dirty L1 data cache lines matching the VA>
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D-cache, 2nd level cache, MMU

3.1.15. th.icache.iall

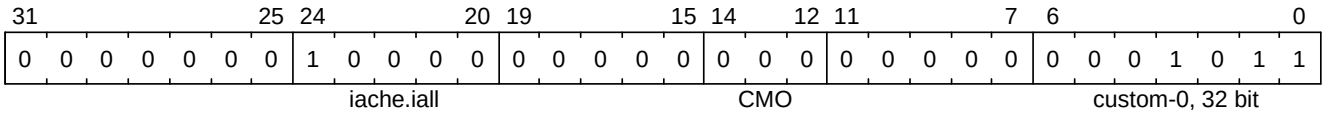
Synopsis

Invalidate all I-cache

Mnemonic

th.icache.iall

Encoding



Description

This instruction invalidates all cache lines of the instruction cache on the local hart. Dirty cache lines will not be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<invalidate all instruction cache lines of the local hart>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	I-cache

3.1.16. th.icache.ialls

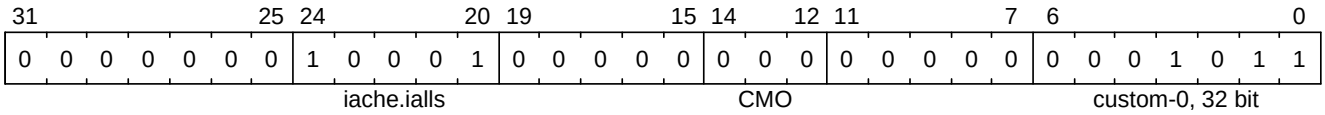
Synopsis

Invalidate all I-cache on all harts

Mnemonic

th.icache.ialls

Encoding



Description

This instruction invalidates all cache lines of the instruction cache on all harts (using broadcasting). Dirty cache lines will not be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<invalidate all instruction cache lines on all harts>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	I-cache

3.1.17. th.icache.ipa

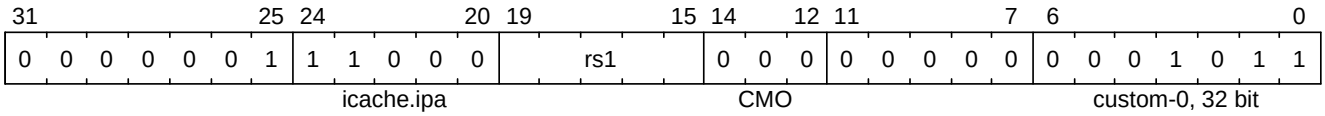
Synopsis

Invalidate I-cache at PA

Mnemonic

th.icache.ipa *rs1*

Encoding



Description

This instruction invalidates the cache lines that match the specified physical address in the I-cache. Dirty cache lines will not be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<invalidate all instruction cache lines matching the PA>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	I-cache

3.1.18. th.icache.iva

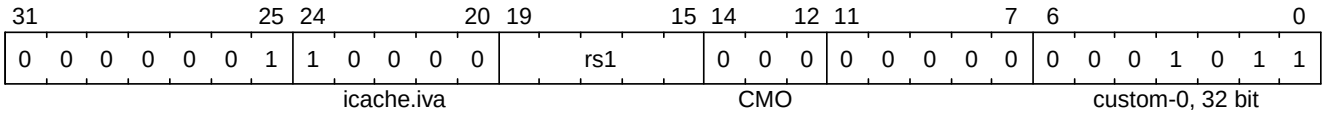
Synopsis

Invalidate I-cache at VA

Mnemonic

th.icache.iva *rs1*

Encoding



Description

This instruction invalidates the cache lines that match the specified virtual address in the I-cache. Dirty cache lines will not be written back to the next-level storage.

Operation

```
<invalidate all instruction cache lines matching the VA>
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	I-cache, MMU

3.1.19. th.l2cache.call

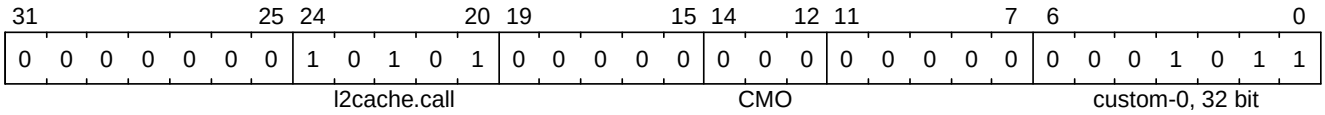
Synopsis

Clean all L2 cache

Mnemonic

th.l2cache.call

Encoding



Description

This instruction cleans all cache lines of the L2 cache. Dirty cache lines will be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<write back all dirty L2 cache lines>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D/I-cache, 2nd level cache

3.1.20. th.l2cache.ciall

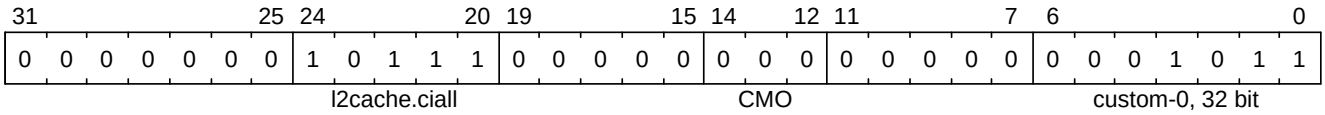
Synopsis

Clean & invalidate all L2 cache

Mnemonic

th.l2cache.ciall

Encoding



Description

This instruction cleans and invalidates all cache lines of the L2 cache. Dirty cache lines will be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<write back all dirty L2 cache lines>
<invalidate all L2 cache lines>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D/I-cache, 2nd level cache

3.1.21. th.l2cache.iall

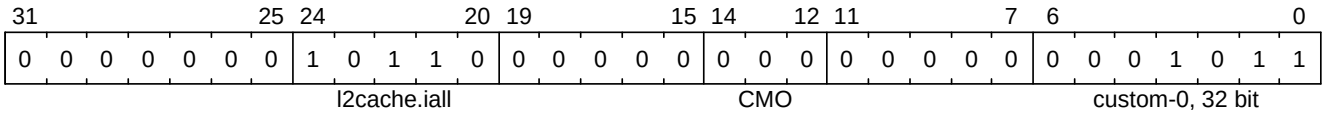
Synopsis

Invalidate all L2 cache

Mnemonic

th.l2cache.iall

Encoding



Description

This instruction invalidates all cache lines of the L2 cache. Dirty cache lines will not be written back to the next-level storage.

Operation

```
if (priv_level == U)
{
  <raise illegal instruction exception>
}

<invalidate all L2 cache lines>
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension	HW requirements
XTheadCmo (Chapter 3)	D/I-cache, 2nd level cache

Chapter 4. Multi-core synchronization instructions (XTheadSync)

The `XTheadSync` extension is `stable`.

The `XTheadSync` ISA extension provides multi-core synchronization instructions.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
Y	Y	th.sfence.vmas *rs1*, *rs2*	Invalidate TLB on all harts
Y	Y	th.sync	Synchronization barrier
Y	Y	th.sync.s	Synchronization barrier
Y	Y	th.sync.i	Synchronization pipeline flush
Y	Y	th.sync.is	Synchronization barrier and pipeline flush

4.1. Instructions

4.1.1. th.sfence.vmas

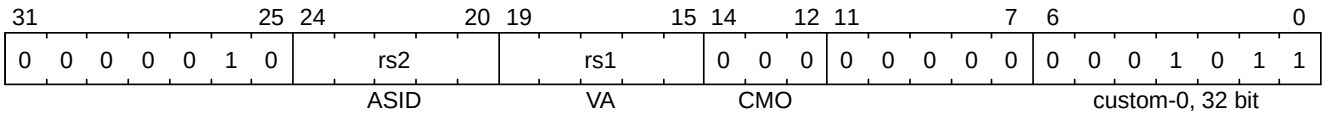
Synopsis

Invalidate TLB (page table cache) on all harts via broadcasting.

Mnemonic

th.sfence.vmas *rs1*, *rs2*

Encoding



Description

This instruction invalidates the TLB (page table cache) on all harts via broadcasting. The register *rs1* holds the virtual address (VA) and *rs2* holds the address space identifier (ASID) of the TLB entry that will be invalidated on all harts via broadcasting.

An operand that is zero is interpreted as match-all. E.g. if *rs2* is zero, then all TLB entries that match the VA in *rs1* are invalidated on all harts via broadcasting. Consequently, if both operands, *rs1* and *rs2*, are zero, then all TLB entries are invalidated on all harts via broadcasting.

Operation

```
if (priv_level == U)
```

```
{
  <raise illegal instruction exception>
}

if _rs1_ != 0
{
  va := _rs1_
}
else
{
  va := _MATCH_ALL_VA_
}

if _rs2_ != 0
{
  asid := _rs2_
}
else
{
  asid := _MATCH_ALL_ASID_
}

msg := encode_invalidate_tlb(va, asid)
broadcast_to_all_harts(msg)
```

Permission

This instruction can be executed in all privilege levels higher than `U` mode. Attempts to execute this instruction in `U` mode raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadSync ([xtheasync])

4.1.2. th.sync

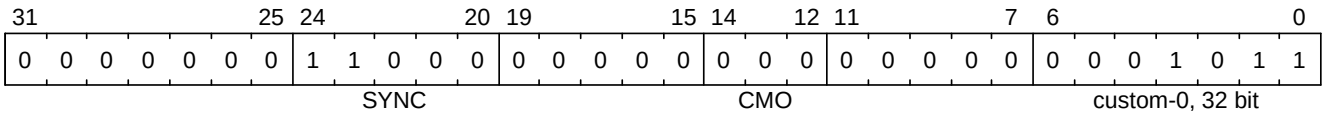
Synopsis

Ensures that all preceding instructions retire earlier than this instruction and all subsequent instructions retire later than this instruction.

Mnemonic

th.sync

Encoding



Description

This instruction ensures that all preceding instructions retire earlier than this instruction and all subsequent instructions retire later than this instruction. It is stricter than standard RISC-V fence instruction. Fence only influences the order of load/store instructions while th.sync influences all the instructions, including all explicit memory accesses and cache operations.

Operation

```
out_of_order_barrier()
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadSync ([xtheasync])

4.1.3. th.sync.s

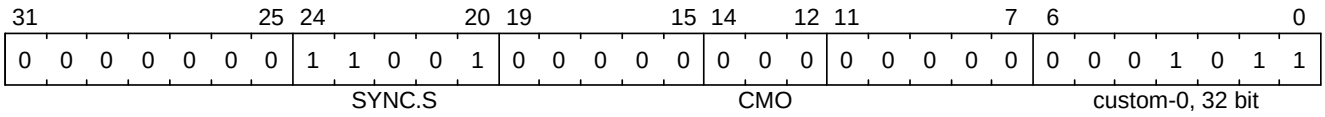
Synopsis

Ensures that all preceding instructions retire earlier than this instruction and all subsequent instructions retire later than this instruction.

Mnemonic

th.sync.s

Encoding



Description

This instruction has the same function with th.sync.

Operation

```
out_of_order_barrier()
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadSync ([xtheasync])

4.1.4. th.sync.i

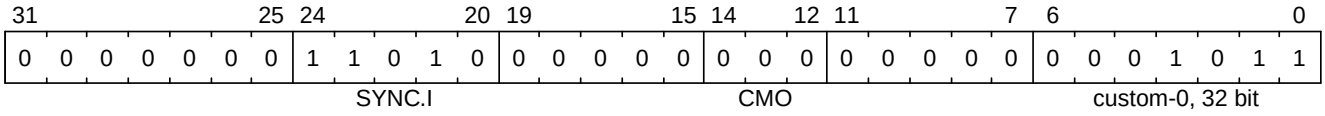
Synopsis

Ensures that all preceding instructions retire earlier than this instruction and all subsequent instructions retire later than this instruction and clears the pipeline when this instruction retires.

Mnemonic

th.sync.i

Encoding



Description

Besides the synopsis of th.sync, this instruction flushes the pipeline of current hart which means all subsequent instructions should be re-fectched after this instruction retires. This instruction is the only mechanism to ensure that all explicit memory accesses or cache operations visible to a hart will also be visible to its instruction fetches.

Operation

```
out_of_order_barrier()
pipeline_flush()
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadSync ([xtheasync])

4.1.5. th.sync.is

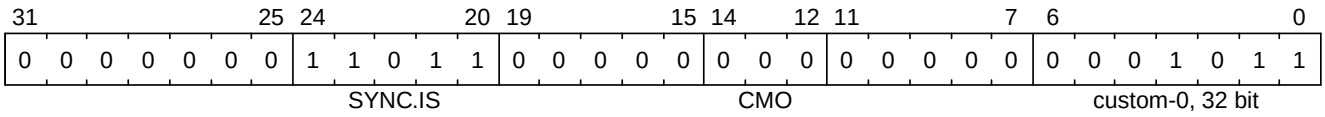
Synopsis

Ensures that all preceding instructions retire earlier than this instruction and all subsequent instructions retire later than this instruction and clears the pipeline when this instruction.

Mnemonic

th.sync.is

Encoding



Description

This instruction has the same function with th.sync.i.

Operation

```
out_of_order_barrier()
pipeline_flush()
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadSync ([xtheasync])

Chapter 5. Address calculation instructions (XTheadBa)

The `XTheadBa` extension is `stable`.

The `XTheadBa` ISA extension provides bitmanipulation instructions for address calculation.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
Y	Y	th.addsl *rd*, *rs1*, *rs2*, *imm2*	Add shifted operand

5.1. Instructions

5.1.1. th.addsl

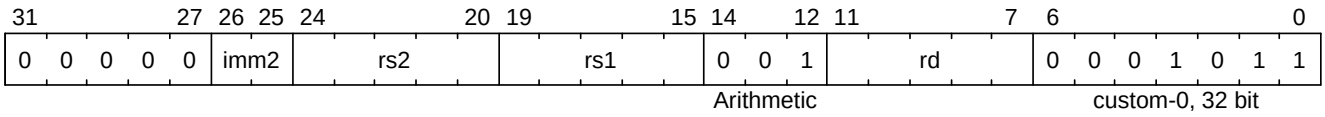
Synopsis

Add a shifted operand to a second operand.

Mnemonic

th.addsl *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This operation adds the shifted operand (*rs2* << *imm2*) with *rs1*.

Operation

```
reg[rd] := reg[rs1] + (reg[rs2] << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBa (Chapter 5)

Chapter 6. Basic bit-manipulation (XTheadBb)

The `XTheadBb` extension is `stable`.

The `XTheadBb` ISA extension provides conditional basic bit-manipulation instructions.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
Y	Y	th.srri *rd*, *rs1*, *imm6*	Cyclic right shift
N	Y	th.srriw *rd*, *rs1*, *imm5*	Cyclic right shift on word operand
Y	Y	th.ext *rd*, *rs1*, *imm1*, *imm2*	Extract and sign-extend bits
Y	Y	th.extu *rd*, *rs1*, *imm1*, *imm2*	Extract and zero-extend bits
Y	Y	th.ff0 *rd*, *rs1*	Find first '0'-bit
Y	Y	th.ff1 *rd*, *rs1*	Find first '1'-bit
Y	Y	th.rev *rd*, *rs1*	Reverse byte order
N	Y	th.revw *rd*, *rs1*	Reverse byte order of word operand
Y	Y	th.tstnbz *rd*, *rs1*	Test for NUL bytes

6.1. Instructions

6.1.1. th.srri

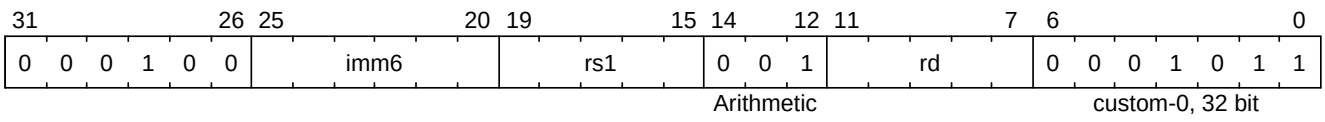
Synopsis

Perform a cyclic right shift.

Mnemonic

th.srri *rd*, *rs1*, *imm6*

Encoding



Description

This operation rotates the contents of *rs1* by *imm6* bits and stores the result in *rd*.

Operation

```
if (xlen == 32)
  imm6 &= 0x1f
```

```
reg[rd] := (reg[rs1] >> imm6) | (reg[rs1] << (xlen - imm6))
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBb (Chapter 6)

6.1.2. th.srriw

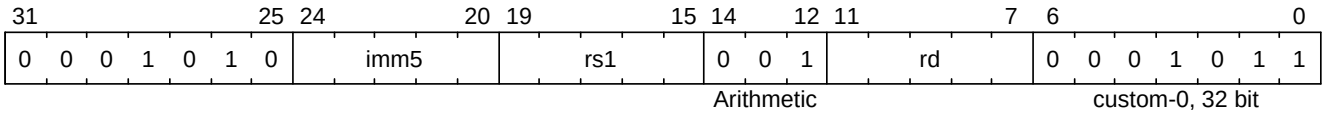
Synopsis

Perform a cyclic right shift on word operand.

Mnemonic

th.srriw *rd*, *rs1*, *imm5*

Encoding



Description

This operation rotates the contents of the 32-bit value in *rs1* by *imm5* bits and stores the result in *rd*.

Operation

```
data := zext.w(reg[rs1])
reg[rd] := (data >> imm5) | (data << (32 - imm5))
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBb (Chapter 6)

6.1.3. th.ext

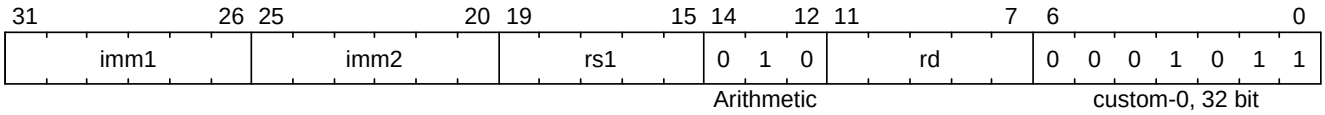
Synopsis

Extract and sign-extend bits.

Mnemonic

th.ext *rd*, *rs1*, *imm1*, *imm2*

Encoding



Description

This operation extract the bits *imm1*..*imm2* from register *rs1*, sign-extends the value, and stores the result in *rd*.

Operation

```
reg[rd] := sign_extend(reg[rs1][imm1:imm2])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBb (Chapter 6)

6.1.4. th.extu

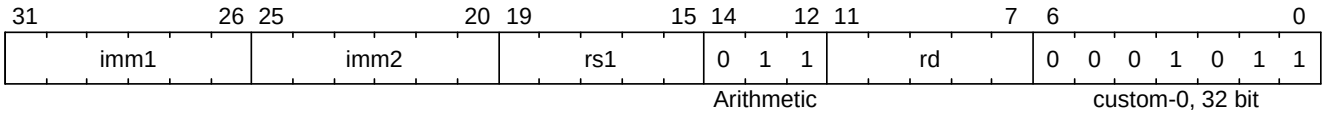
Synopsis

Extract and zero-extend bits.

Mnemonic

th.extu *rd*, *rs1*, *imm1*, *imm2*

Encoding



Description

This operation extract the bits *imm1*..*imm2* from register *rs1*, zero-extends the value, and stores the result in *rd*.

Operation

```
reg[rd] := zero_extend(reg[rs1][imm1:imm2])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBb (Chapter 6)

6.1.5. th.ff0

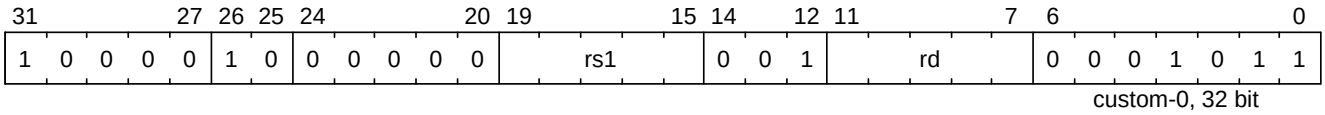
Synopsis

Find first '0'-bit

Mnemonic

th.ff0 *rd*, *rs1*

Encoding



Description

Finds the first bit with the value of '0' from the highest bit of *rs1* and writes the index back into register *rd*. If the highest bit of *rs1* is '0', the result '0' is returned. If all the bits in *rs1* are '1', the result '64' is returned.

Operation

```
for i=xlen..0:
  if reg[rs1][i] == 0:
    break;
reg[rd] = (xlen - 1) - i
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBb (Chapter 6)

6.1.6. th.ff1

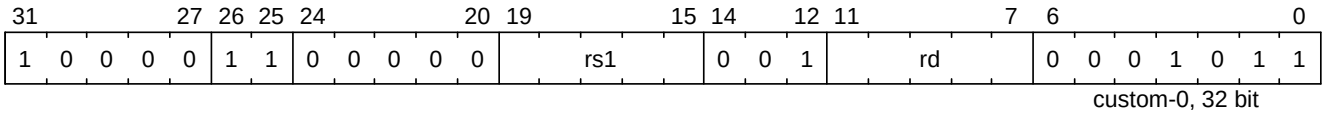
Synopsis

Find first '1'-bit

Mnemonic

th.ff1 *rd*, *rs1*

Encoding



Description

Finds the first bit with the value of '1' from the highest bit of *rs1* and writes the index back into register *rd*. If the highest bit of *rs1* is '1', the result '0' is returned. If all the bits in *rs1* are '1', the result '64' is returned.

Operation

```
for i=xlen..0:
  if reg[rs1][i] == 1:
    break;
reg[rd] = (xlen - 1) - i
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBb (Chapter 6)

6.1.7. th.rev

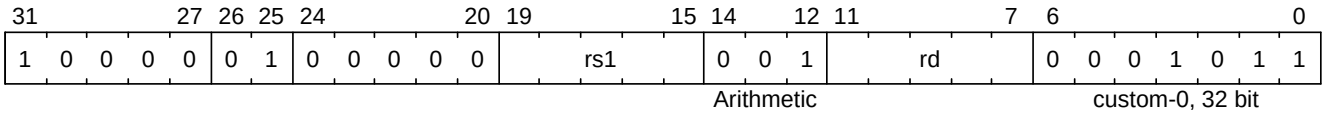
Synopsis

Reverse the byte order.

Mnemonic

th.rev *rd, rs1*

Encoding



Description

This operation reverses the byte order of the value in *rs1* and stores the result in *rd*.

Operation

```
for i=0..(xlen/8-1):
  j := xlen/8 - 1 - i
  tmp[i] := reg[rs1][j]
reg[rd] := tmp
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBb (Chapter 6)

6.1.8. th.revw

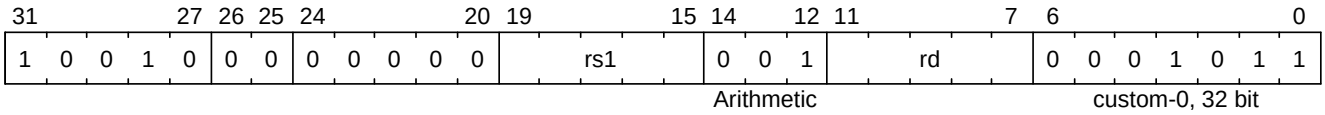
Synopsis

Reverse the byte order of a word operand.

Mnemonic

th.revw *rd*, *rs1*

Encoding



Description

This operation reverses the byte order of the 32-bit value in *rs1* and stores the result in *rd*.

Operation

```
for i=0..3:
  j := 3 - i
  tmp[i] := reg[rs1][j]
reg[rd] := tmp
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBb (Chapter 6)

6.1.9. th.tstnbz

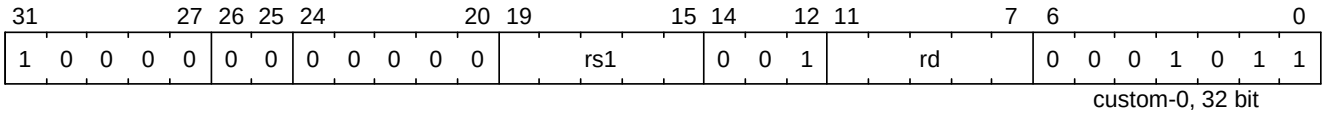
Synopsis

Test for NUL bytes.

Mnemonic

th.tstnbz *rd*, *rs1*

Encoding



Description

Tests each byte in register *rs1* for equality with 0. If a byte is 0, then the corresponding byte in register *rd* will be set to 0xff. Otherwise, the corresponding byte in register *rd* will be set to 0.

Operation

```
for i=0..(xlen/8-1):
  if reg[rs1][i] == 0:
    reg[rd][i] := 0xff
  else
    reg[rd][i] := 0
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBb (Chapter 6)

Chapter 7. Single-bit instructions (XTheadBs)

The `XTheadBs` extension is `stable`.

The `XTheadBs` ISA extension provides instructions to access a single bit in a register.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
Y	Y	th.tst *rd*, *rs1*, *imm6*	Test bit

7.1. Instructions

7.1.1. th.tst

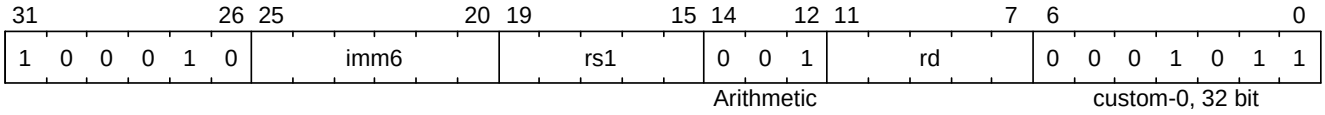
Synopsis

Tests if a single bit is set.

Mnemonic

th.tst *rd*, *rs1*, *imm6*

Encoding



Description

This instruction tests if a single bit is set. If so, *rd* will be set to 1. Otherwise, *rd* will be set to 0.

Operation

```
if (reg[rs1] & (1 << imm6))
  rd := 1
else
  rd := 0
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadBs (Chapter 7)

Chapter 8. Conditional move (XTheadCondMov)

The `XTheadCondMov` extension is `stable`.

The `XTheadCondMov` ISA extension provides conditional move instructions.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
Y	Y	th.mveqz *rd*, *rs1*, *rs2*	Move if equal zero
Y	Y	th.mvnez *rd*, *rs1*, *rs2*	Move if not equal zero

8.1. Instructions

8.1.1. th.mveqz

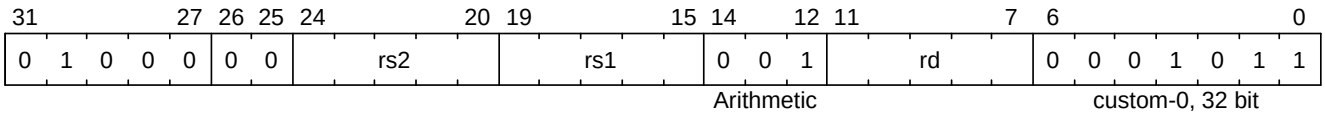
Synopsis

Move if equal zero.

Mnemonic

th.mveqz *rd*, *rs1*, *rs2*

Encoding



Description

This instruction moves the content of register *rs1* into *rd* if the content of *rs2* is 0x0. Otherwise, the value of *rd* does not change.

Operation

```
if (reg[rs2] == 0x0)
  reg[rd] := reg[rs1]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadCondMov (Chapter 8)

8.1.2. th.mvnez

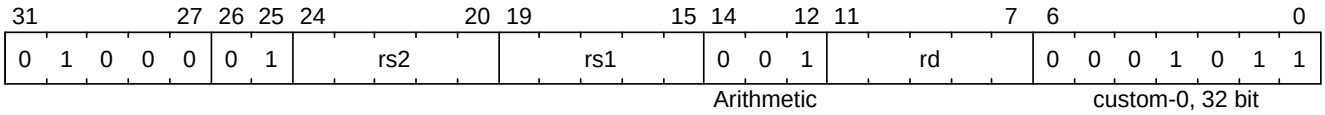
Synopsis

Move if not equal zero.

Mnemonic

th.mvnez *rd*, *rs1*, *rs2*

Encoding



Description

This instruction moves the content of register *rs1* into *rd* if the content of *rs2* is not 0x0. Otherwise, the value of *rd* does not change.

Operation

```
if (reg[rs2] != 0x0)
  reg[rd] := reg[rs1]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadCondMov (Chapter 8)

Chapter 9. Indexed memory operations (XTheadMemIdx)

The `XTheadMemIdx` extension is `stable`.

The `XTheadMemIdx` ISA extension provides indexed memory operations. for GP registers.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
Y	Y	th.lbia *rd*, (*rs1*), *imm5*, *imm2*	Load indexed byte
Y	Y	th.lbib *rd*, (*rs1*), *imm5*, *imm2*	Load indexed byte
Y	Y	th.lbuia *rd*, (*rs1*), *imm5*, *imm2*	Load indexed unsigned byte
Y	Y	th.lbuib *rd*, (*rs1*), *imm5*, *imm2*	Load indexed unsigned byte
Y	Y	th.lhia *rd*, (*rs1*), *imm5*, *imm2*	Load indexed half-word
Y	Y	th.lhib *rd*, (*rs1*), *imm5*, *imm2*	Load indexed half-word
Y	Y	th.lhuia *rd*, (*rs1*), *imm5*, *imm2*	Load indexed unsigned half-word
Y	Y	th.lhuib *rd*, (*rs1*), *imm5*, *imm2*	Load indexed unsigned half-word
Y	Y	th.lwia *rd*, (*rs1*), *imm5*, *imm2*	Load indexed word
Y	Y	th.lwib *rd*, (*rs1*), *imm5*, *imm2*	Load indexed word
N	Y	th.lwuia *rd*, (*rs1*), *imm5*, *imm2*	Load indexed unsigned word
N	Y	th.lwuib *rd*, (*rs1*), *imm5*, *imm2*	Load indexed unsigned word
N	Y	th.ldia *rd*, (*rs1*), *imm5*, *imm2*	Load indexed double-word
N	Y	th.ldib *rd*, (*rs1*), *imm5*, *imm2*	Load indexed double-word
Y	Y	th.sbia *rd*, (*rs1*), *imm5*, *imm2*	Store indexed byte
Y	Y	th.sbib *rd*, (*rs1*), *imm5*, *imm2*	Store indexed byte
Y	Y	th.shia *rd*, (*rs1*), *imm5*, *imm2*	Store indexed half-word
Y	Y	th.shib *rd*, (*rs1*), *imm5*, *imm2*	Store indexed half-word
Y	Y	th.swia *rd*, (*rs1*), *imm5*, *imm2*	Store indexed word
Y	Y	th.swib *rd*, (*rs1*), *imm5*, *imm2*	Store indexed word
N	Y	th.sdia *rd*, (*rs1*), *imm5*, *imm2*	Store indexed double-word
N	Y	th.sdib *rd*, (*rs1*), *imm5*, *imm2*	Store indexed double-word
Y	Y	th.lrb *rd*, *rs1*, *rs2*, *imm2*	Load indexed byte
Y	Y	th.lrbu *rd*, *rs1*, *rs2*, *imm2*	Load indexed unsigned byte
Y	Y	th.lrh *rd*, *rs1*, *rs2*, *imm2*	Load indexed half-word
Y	Y	th.lrhu *rd*, *rs1*, *rs2*, *imm2*	Load indexed unsigned half-word

RV32	RV64	Mnemonic	Instruction
Y	Y	th.lrw *rd*, *rs1*, *rs2*, *imm2*	Load indexed word
N	Y	th.lrwu *rd*, *rs1*, *rs2*, *imm2*	Load indexed unsigned word
N	Y	th.lrd *rd*, *rs1*, *rs2*, *imm2*	Load indexed double-word
Y	Y	th.srb *rd*, *rs1*, *rs2*, *imm2*	Store indexed byte
Y	Y	th.srh *rd*, *rs1*, *rs2*, *imm2*	Store indexed half-word
Y	Y	th.srw *rd*, *rs1*, *rs2*, *imm2*	Store indexed word
N	Y	th.srd *rd*, *rs1*, *rs2*, *imm2*	Store indexed double-word
Y	Y	th.lurb *rd*, *rs1*, *rs2*, *imm2*	Load unsigned indexed byte
Y	Y	th.lurbu *rd*, *rs1*, *rs2*, *imm2*	Load unsigned indexed unsigned byte
Y	Y	th.lurh *rd*, *rs1*, *rs2*, *imm2*	Load unsigned indexed half-word
Y	Y	th.lurhu *rd*, *rs1*, *rs2*, *imm2*	Load unsigned indexed unsigned half-word
Y	Y	th.lurw *rd*, *rs1*, *rs2*, *imm2*	Load unsigned indexed word
N	Y	th.lurwu *rd*, *rs1*, *rs2*, *imm2*	Load unsigned indexed unsigned word
N	Y	th.lurd *rd*, *rs1*, *rs2*, *imm2*	Load unsigned indexed double-word
Y	Y	th.surb *rd*, *rs1*, *rs2*, *imm2*	Store unsigned indexed byte
Y	Y	th.surh *rd*, *rs1*, *rs2*, *imm2*	Store unsigned indexed half-word
Y	Y	th.surw *rd*, *rs1*, *rs2*, *imm2*	Store unsigned indexed word
N	Y	th.surd *rd*, *rs1*, *rs2*, *imm2*	Store unsigned indexed double-word

9.1. Instructions

9.1.1. th.lbia

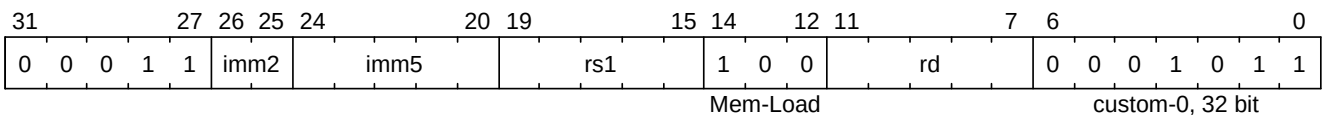
Synopsis

Load indexed byte, increment address after loading.

Mnemonic

th.lbia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction loads a sign extended 8-bit value into the GP register *rd* from the address *rs1*. After the load, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rd := sign_extend(mem[rs1])
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LB` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.2. th.lbib

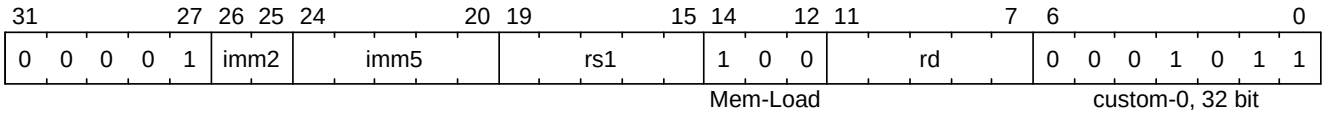
Synopsis

Load indexed byte, increment address before loading.

Mnemonic

th.lbib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction loads a sign extended 8-bit value into the GP register *rd* from the (incremented) address *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
rd := sign_extend(mem[rs1])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LB` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.3. th.lbuia

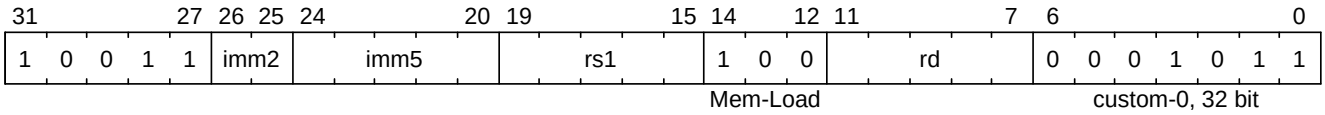
Synopsis

Load indexed unsigned byte, increment address after loading.

Mnemonic

th.lbuia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction loads a zero extended 8-bit value into the GP register *rd* from the address *rs1*. After the load, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rs := zero_extend(mem[rs1])
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LBU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.4. th.lbuib

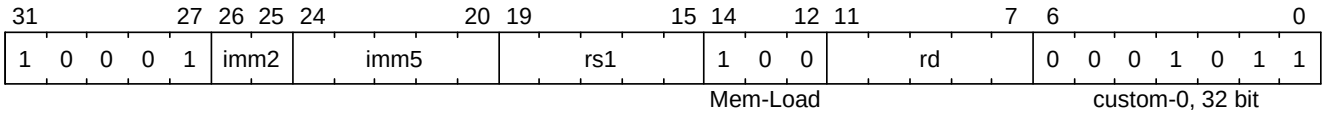
Synopsis

Load indexed unsigned byte, increment address before loading.

Mnemonic

th.lbuib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction loads a zero extended 8-bit value into the GP register *rd* from the (incremented) address *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
rd := zero_extend(mem[rs1])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LBU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.5. th.lhia

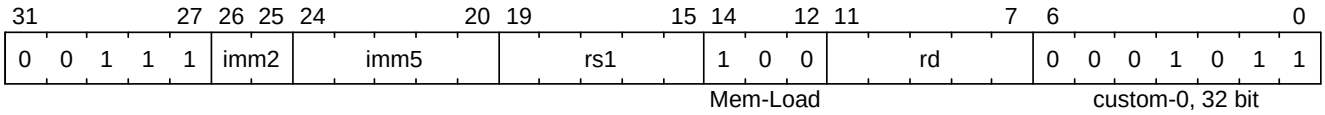
Synopsis

Load indexed half-word, increment address after loading.

Mnemonic

th.lhia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction loads a sign extended 16-bit value into the GP register *rd* from the address *rs1*. After the load, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rd := sign_extend(mem[rs1+1:rs1])
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LH` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.6. th.lhib

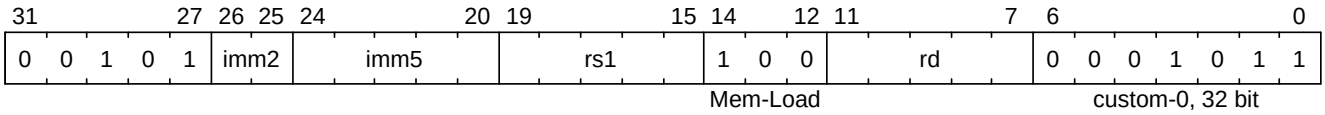
Synopsis

Load indexed half-word, increment address before loading.

Mnemonic

th.lhib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction loads a sign extended 16-bit value into the GP register *rd* from the (incremented) address *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
rd := sign_extend(mem[rs1+1:rs1])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LH` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.7. th.lhuia

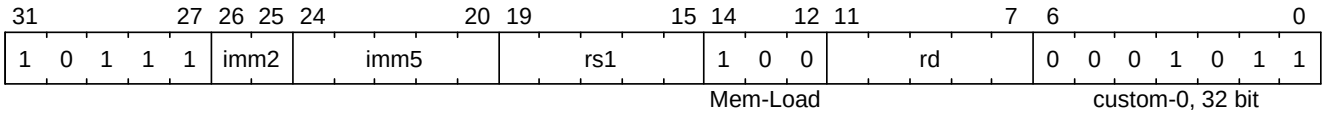
Synopsis

Load indexed unsigned half-word, increment address after loading.

Mnemonic

th.lhuia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction loads a zero extended 16-bit value into the GP register *rd* from the address *rs1*. After the load, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rd := zero_extend(mem[rs1+1:rs1])
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LHU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.8. th.lhuib

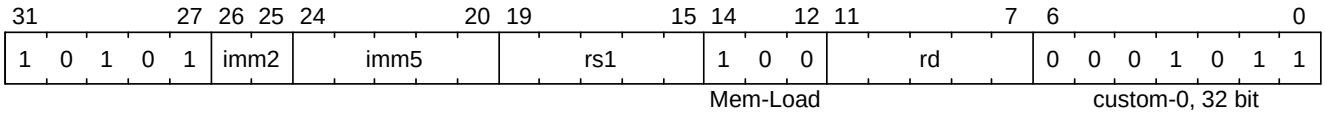
Synopsis

Load indexed unsigned half-word, increment address before loading.

Mnemonic

th.lhuib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction loads a zero extended 16-bit value into the GP register *rd* from the (incremented) address *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
rd := zero_extend(mem[rs1+1:rs1])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LHU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.9. th.lwia

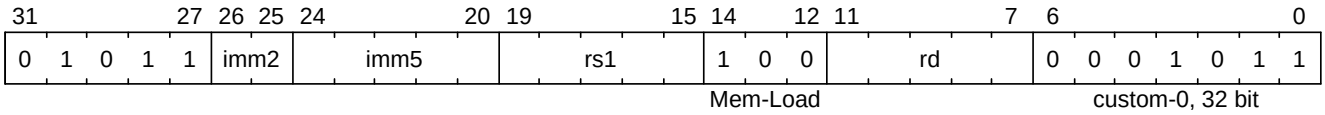
Synopsis

Load indexed word, increment address after loading.

Mnemonic

th.lwia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction loads a sign extended 32-bit value into the GP register *rd* from the address *rs1*. After the load, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rd := sign_extend(mem[rs1+3:rs1])
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LW` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.10. th.lwib

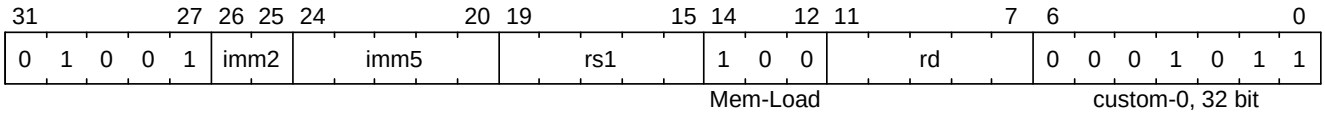
Synopsis

Load indexed word, increment address before loading.

Mnemonic

th.lwib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction loads a sign extended 32-bit value into the GP register *rd* from the (incremented) address *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
rd := sign_extend(mem[rs1+3:rs1])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LW` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.11. th.lwuia

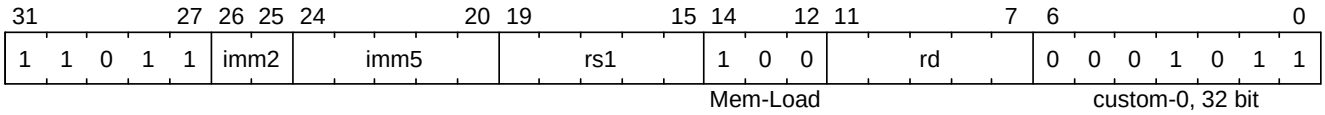
Synopsis

Load indexed unsigned word, increment address after loading.

Mnemonic

th.lwuia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction loads a zero extended 32-bit value into the GP register *rd* from the address *rs1*. After the load, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rd := zero_extend(mem[rs1+3:rs1])
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LWU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.12. th.lwuib

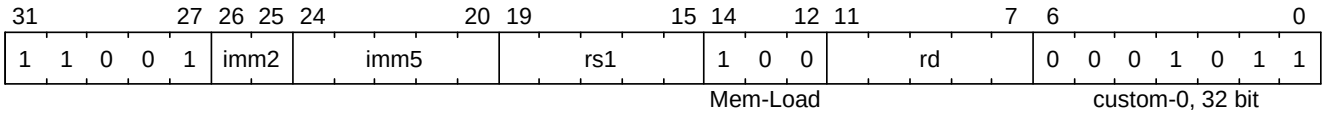
Synopsis

Load indexed unsigned word, increment address before loading.

Mnemonic

th.lwuib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction loads a zero extended 32-bit value into the GP register *rd* from the (incremented) address *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
rd := zero_extend(mem[rs1+3:rs1])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LWU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.13. th.ldia

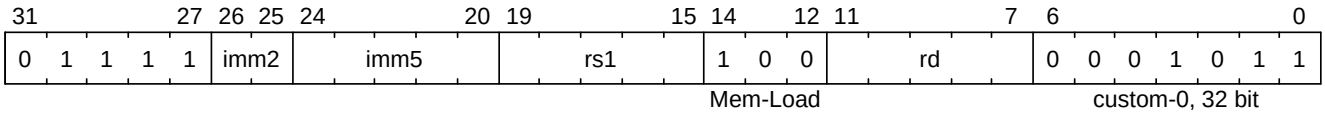
Synopsis

Load indexed double-word, increment address after loading.

Mnemonic

th.ldia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction loads a sign extended 64-bit value into the GP register *rd* from the address *rs1*. After the load, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rd := sign_extend(mem[rs1+7:rs1])
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LD` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.14. th.ldib

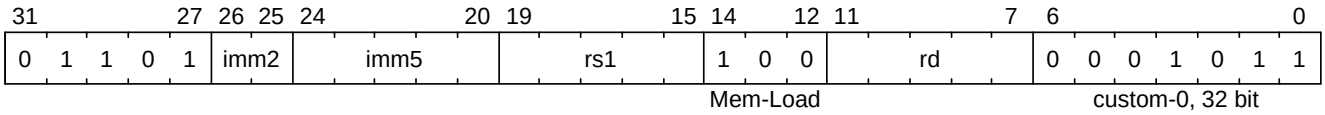
Synopsis

Load indexed double-word, increment address before loading.

Mnemonic

th.lwib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction loads a sign extended 64-bit value into the GP register *rd* from the (incremented) address *rs1*.

The values of *rd* and *rs1* must not be the same.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
rd := sign_extend(mem[rs1+7:rs1])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LD` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.15. th.sbia

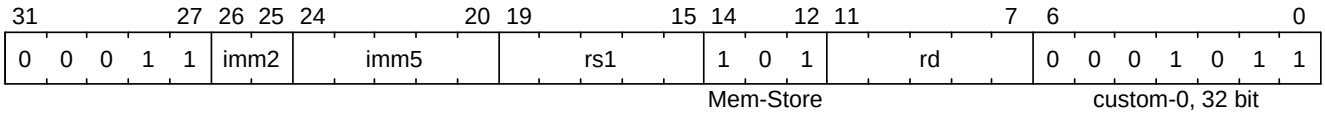
Synopsis

Store indexed byte, increment address after loading.

Mnemonic

th.sbia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction stores an 8-bit value from the GP register *rd* to the address *rs1*. After the store, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

Operation

```
mem[rs1] := rd
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SB` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.16. th.sbib

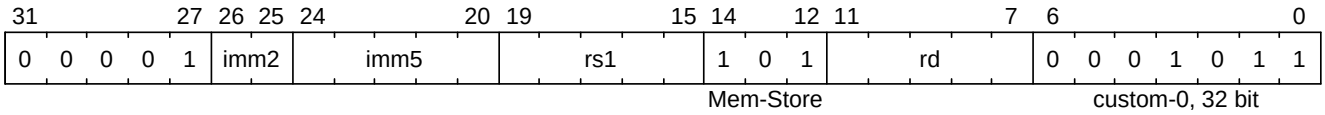
Synopsis

Store indexed byte, increment address before loading.

Mnemonic

th.sbib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction stores an 8-bit value from the GP register *rd* to the (incremented) address *rs1*.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
mem[rs1] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SB` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.17. th.shia

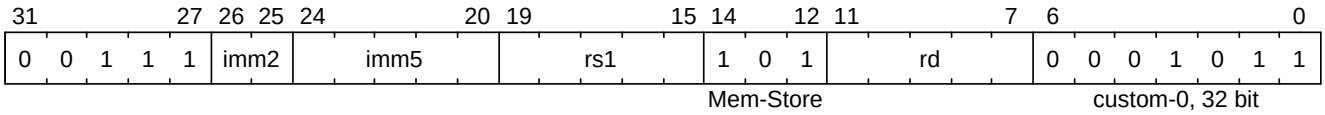
Synopsis

Store indexed half-word, increment address after loading.

Mnemonic

th.shia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction stores an 16-bit value from the GP register *rd* to the address *rs1*. After the store, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

Operation

```
mem[rs1+1:rs1] := rd
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SH` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.18. th.shib

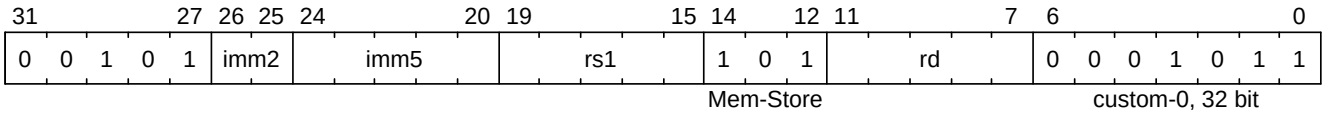
Synopsis

Store indexed half-word, increment address before loading.

Mnemonic

th.shib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction stores an 16-bit value from the GP register *rd* to the (incremented) address *rs1*.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
mem[rs1+1:rs1] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SH` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.19. th.swia

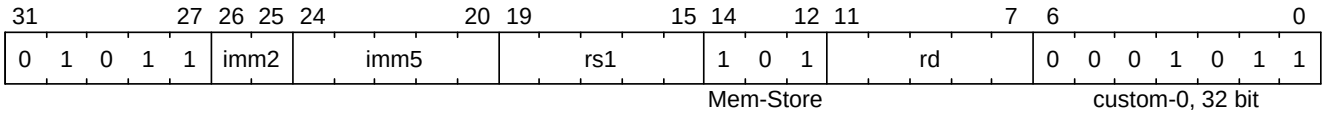
Synopsis

Store indexed word, increment address after loading.

Mnemonic

th.swia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction stores an 32-bit value from the GP register *rd* to the address *rs1*. After the store, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

Operation

```
mem[rs1+3:rs1] := rd
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SW` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.20. th.swib

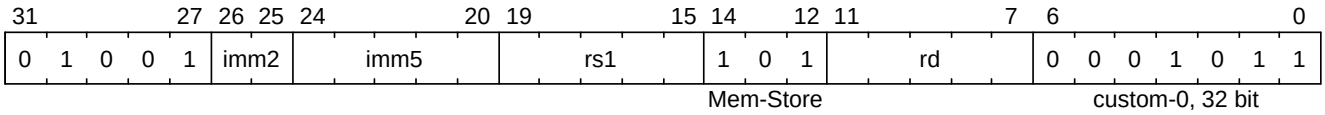
Synopsis

Store indexed word, increment address before loading.

Mnemonic

th.swib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction stores an 32-bit value from the GP register *rd* to the (incremented) address *rs1*.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
mem[rs1+3:rs1] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SW` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.21. th.sdia

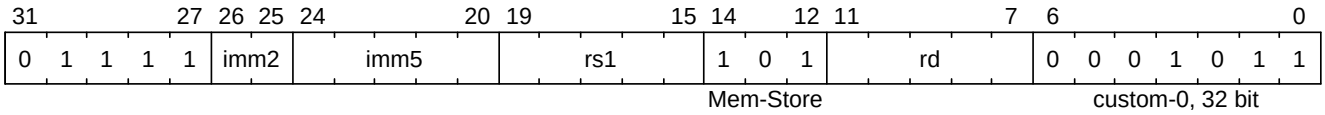
Synopsis

Store indexed double-word, increment address after loading.

Mnemonic

th.sdia *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction stores an 64-bit value from the GP register *rd* to the address *rs1*. After the store, this instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*.

Operation

```
mem[rs1+7:rs1] := rd
rs1 := rs1 + (sign_extend(imm5) << imm2)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SD` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.22. th.sdib

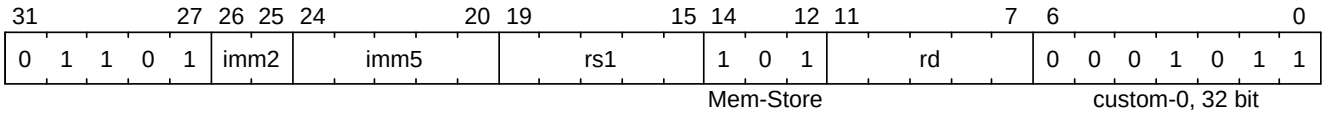
Synopsis

Store indexed double-word, increment address before loading.

Mnemonic

th.sdib *rd*, (*rs1*), *imm5*, *imm2*

Encoding



Description

This instruction increments the value in *rs1* by (sign_extend(*imm5*) << *imm2*) and writes the result back to *rs1*. After the increment of *rs1*, this instruction stores an 64-bit value from the GP register *rd* to the (incremented) address *rs1*.

Operation

```
rs1 := rs1 + (sign_extend(imm5) << imm2)
mem[rs1+7:rs1] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SD` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.23. th.lrb

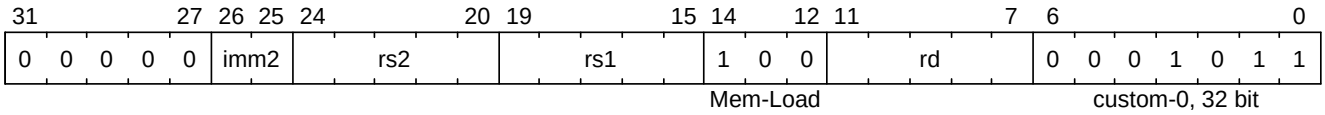
Synopsis

Load indexed byte.

Mnemonic

th.lrb *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a sign extended 8-bit value into the GP register *rd* from the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
rd := sign_extend(mem[addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LB` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.24. th.lrbu

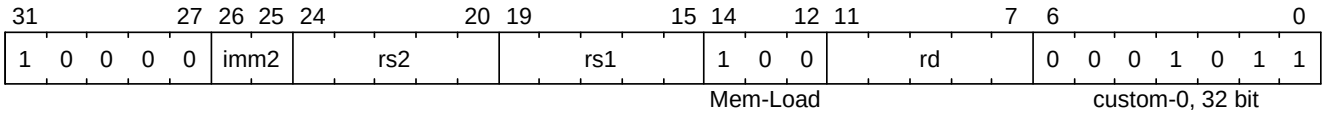
Synopsis

Load indexed unsigned byte.

Mnemonic

th.lrbu *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a zero extended 8-bit value into the GP register *rd* from the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
rd := zero_extend(mem[addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LBU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.25. th.lrh

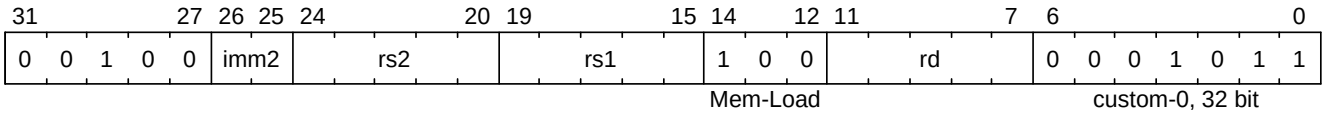
Synopsis

Load indexed half-word.

Mnemonic

th.lrh *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a sign extended 16-bit value into the GP register *rd* from the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
rd := sign_extend(mem[addr+1:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LH` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.26. th.lrhu

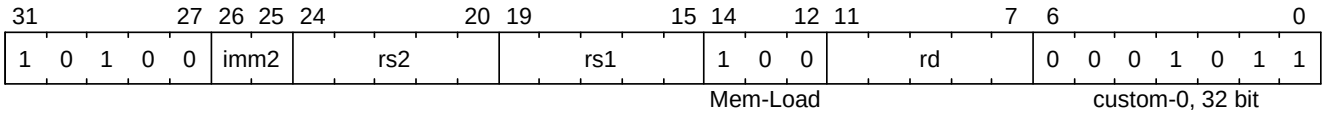
Synopsis

Load indexed unsigned half-word.

Mnemonic

th.lrhu *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a zero extended 16-bit value into the GP register *rd* from the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
rd := zero_extend(mem[addr+1:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LHU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.27. th.lrw

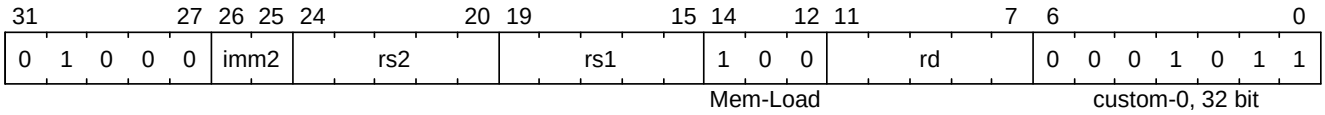
Synopsis

Load indexed word.

Mnemonic

th.lrw *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a sign extended 32-bit value into the GP register *rd* from the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
rd := sign_extend(mem[addr+3:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LW` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.28. th.lrwu

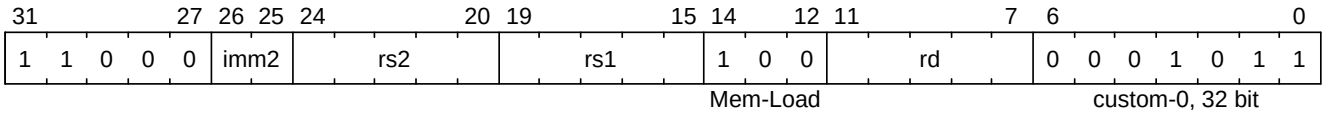
Synopsis

Load indexed unsigned word.

Mnemonic

th.lrwu *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a zero extended 32-bit value into the GP register *rd* from the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
rd := zero_extend(mem[addr+3:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LWU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.29. th.lrd

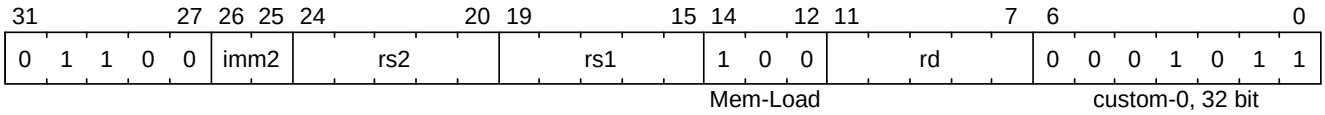
Synopsis

Load indexed word.

Mnemonic

th.lrd *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a sign extended 64-bit value into the GP register *rd* from the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
rd := sign_extend(mem[addr+7:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LD` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.30. th.srb

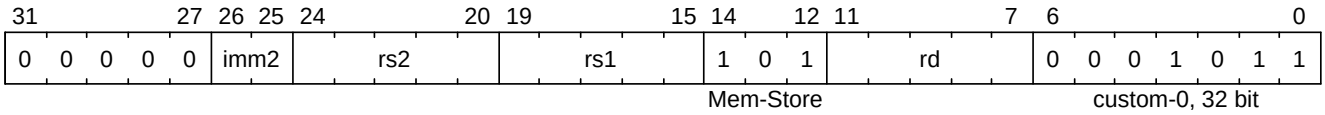
Synopsis

Store indexed byte.

Mnemonic

th.srb *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores an 8-bit value from the GP register *rd* to the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
mem[addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding SB instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.31. th.srh

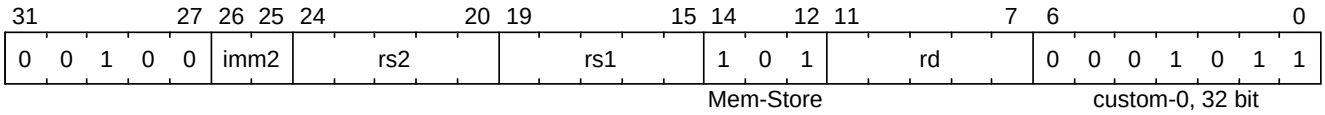
Synopsis

Store indexed half-word.

Mnemonic

th.srh *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores a 16-bit value from the GP register *rd* to the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
mem[addr+1:addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding SH instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.32. th.srw

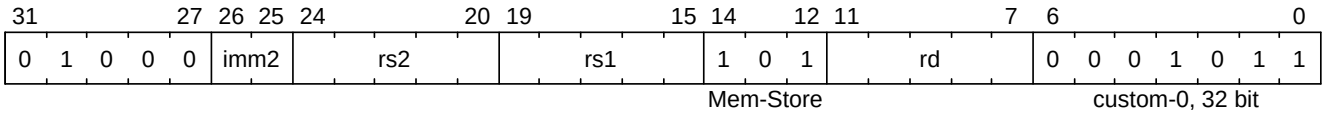
Synopsis

Store indexed word.

Mnemonic

th.srw *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores a 32-bit value from the GP register *rd* to the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
mem[addr+3:addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SW` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.33. th.srd

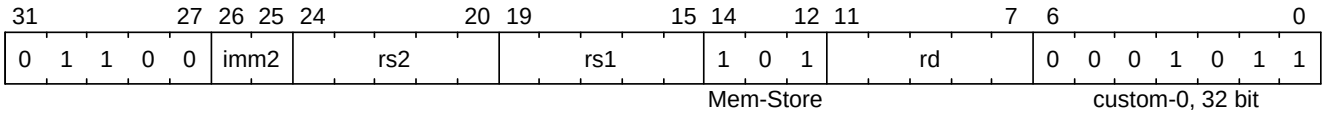
Synopsis

Store indexed double-word.

Mnemonic

th.srd *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores a 64-bit value from the GP register *rd* to the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
mem[addr+7:addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding SD instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.34. th.lurb

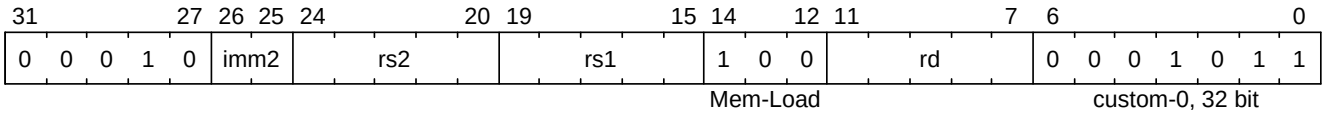
Synopsis

Load unsigned indexed byte.

Mnemonic

th.lurb *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a sign extended 8-bit value into GP register *rd* from the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.lrb` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
rd := sign_extend(mem[addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LB` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.35. th.lurbu

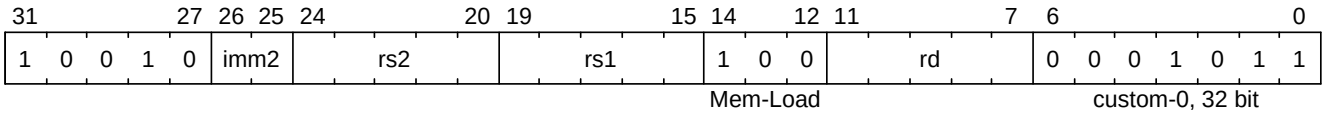
Synopsis

Load unsigned indexed unsigned byte.

Mnemonic

th.lurbu *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a zero extended 8-bit value into GP register *rd* from the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.lrbu` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
rd := zero_extend(mem[addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LBU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.36. th.lurh

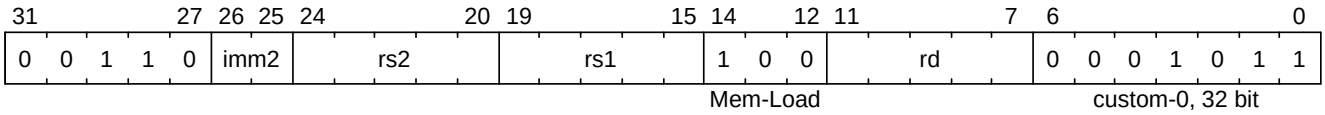
Synopsis

Load unsigned indexed half-word.

Mnemonic

th.lurh *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a sign extended 16-bit value into GP register *rd* from the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.lrh` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
rd := sign_extend(mem[addr+1:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LH` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.37. th.lurhu

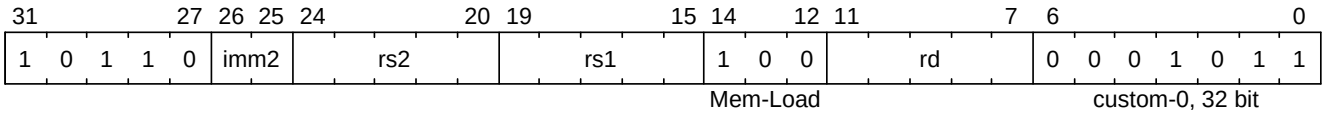
Synopsis

Load unsigned indexed unsigned half-word.

Mnemonic

th.lurhu *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a zero extended 16-bit value into GP register *rd* from the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.lrhu` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
rd := zero_extend(mem[addr+1:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LHU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.38. th.lurw

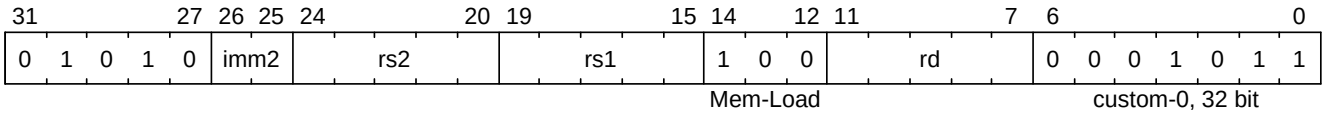
Synopsis

Load unsigned indexed word.

Mnemonic

th.lurw *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a sign extended 32-bit value into GP register *rd* from the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.lrw` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
rd := sign_extend(mem[addr+3:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LW` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.39. th.lurwu

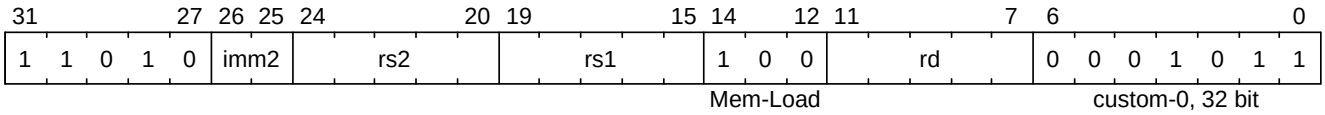
Synopsis

Load unsigned indexed unsigned word.

Mnemonic

th.lurwu *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a zero extended 32-bit value into GP register *rd* from the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.lrwu` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
rd := zero_extend(mem[addr+3:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LWU` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.40. th.lurd

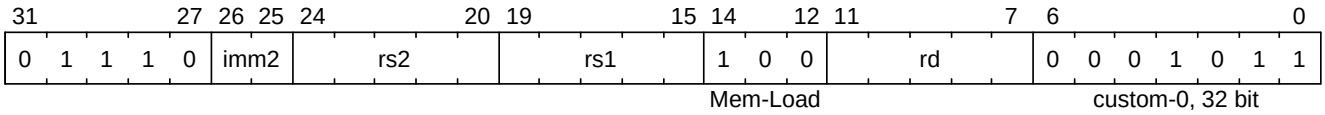
Synopsis

Load unsigned indexed double-word.

Mnemonic

th.lurd *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a sign extended 64-bit value into GP register *rd* from the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.lrd` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
rd := sign_extend(mem[addr+7:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `LD` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.41. th.surb

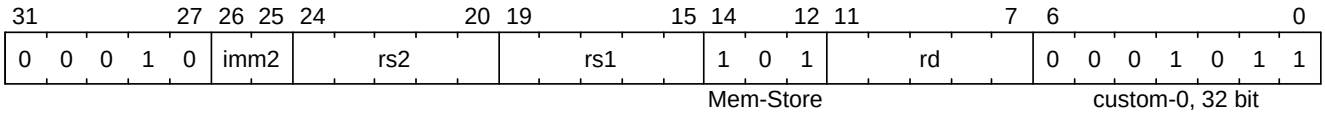
Synopsis

Store unsigned indexed byte.

Mnemonic

th.surb *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores an 8-bit value from the GP register *rd* to the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.srb` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
mem[addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SB` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.42. th.surh

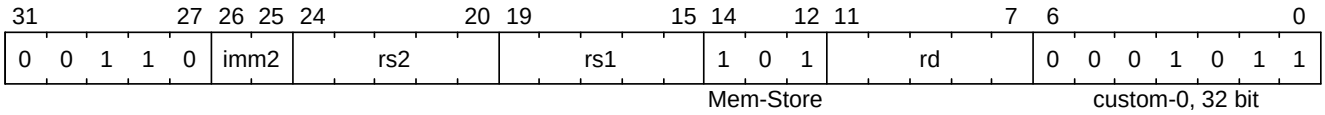
Synopsis

Store unsigned indexed half-word.

Mnemonic

th.surh *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores a 16-bit value from the GP register *rd* to the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.srh` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
mem[addr+1:addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SH` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.43. th.surw

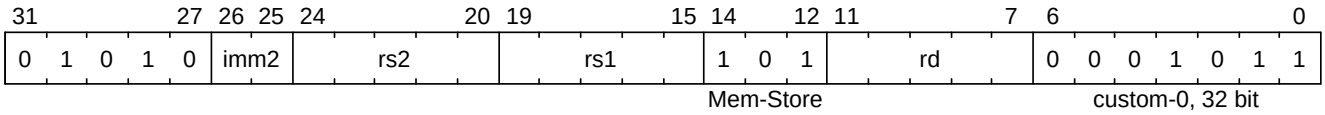
Synopsis

Store unsigned indexed word.

Mnemonic

th.surw *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores a 32-bit value from the GP register *rd* to the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.srw` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
mem[addr+3:addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SW` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

9.1.44. th.surd

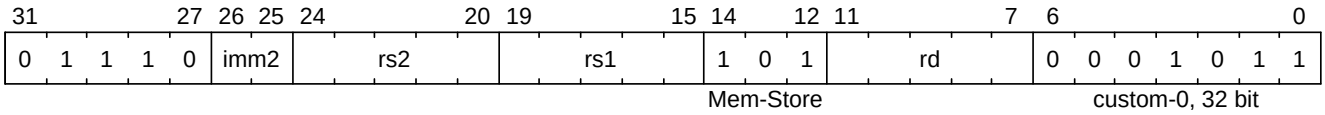
Synopsis

Store unsigned indexed double-word.

Mnemonic

th.surd *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores a 64-bit value from the GP register *rd* to the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.srd` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
mem[addr+7:addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `SD` instruction would trigger.

Included in

Extension
XTheadMemIdx (Chapter 9)

Chapter 10. Two-GPR memory operations (XTheadMemPair)

The `XTheadMemPair` extension is `stable`.

The `XTheadMemPair` ISA extension provides two-GPR memory operations.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
N	Y	th.ldd *rd1*, *rd2*, (*rs1*), *imm2*, 4	Load two 64-bit values
Y	Y	th.lwd *rd1*, *rd2*, (*rs1*), *imm2*, 3	Load two signed 32-bit values
Y	Y	th.lwud *rd1*, *rd2*, (*rs1*), *imm2*, 3	Load two unsigned 32-bit values
N	Y	th.sdd *rd1*, *rd2*, (*rs1*), *imm2*, 4	Store two 64-bit values
Y	Y	th.swd *rd1*, *rd2*, (*rs1*), *imm2*, 3	Store two 32-bit values

10.1. Instructions

10.1.1. th.ldd

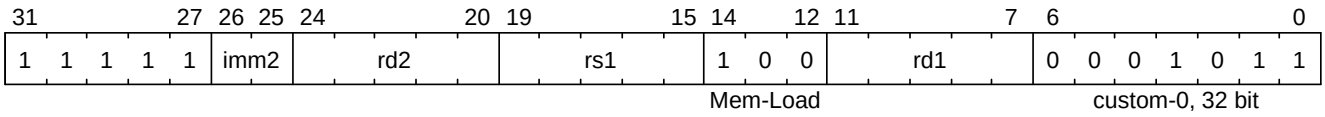
Synopsis

Load two 64-bit values from memory into two GPRs.

Mnemonic

th.ldd *rd1*, *rd2*, (*rs1*), *imm2*, 4

Encoding



Description

This instruction loads two 64-bit values into the two GP registers *rd1* and *rd2* from the address *rs1* + (zero_extend(*imm2*) << 4).

The values of *rd1*, *rd2* and *rs1* must not be the same.

Operation

```
addr := rs1 + (zero_extend(imm2) << 4)
rd1 := mem[addr+7:addr]
rd2 := mem[addr+15:addr+8]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding `LD` instructions would trigger.

Included in

Extension
XTheadMemPair (Chapter 10)

10.1.2. th.lwd

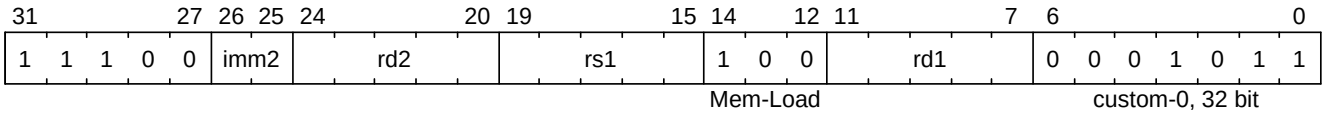
Synopsis

Load two signed 32-bit values from memory into two GPRs.

Mnemonic

th.lwd *rd1*, *rd2*, (*rs1*), *imm2*, 3

Encoding



Description

This instruction loads two signed 32-bit values into the two GP registers *rd1* and *rd2* from the address *rs1* + (zero_extend(*imm2*) << 3).

The values of *rd1*, *rd2* and *rs1* must not be the same.

Operation

```
addr := rs1 + (zero_extend(imm2) << 3)
reg[rd1] := sign_extend(mem[addr+3:addr])
reg[rd2] := sign_extend(mem[addr+7:addr+3])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding `LW` instructions would trigger.

Included in

Extension
XTheadMemPair (Chapter 10)

10.1.3. th.lwud

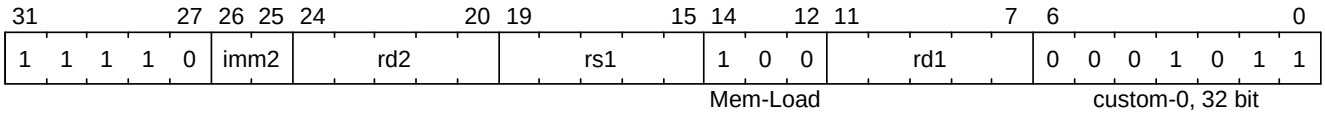
Synopsis

Load two unsigned 32-bit values from memory into two GPRs.

Mnemonic

th.lwud *rd1*, *rd2*, (*rs1*), *imm2*, 3

Encoding



Description

This instruction loads two unsigned 32-bit values into the two GP registers *rd1* and *rd2* from the address *rs1* + (zero_extend(*imm2*) << 3).

The values of *rd1*, *rd2* and *rs1* must not be the same.

Operation

```
addr := rs1 + (zero_extend(imm2) << 3)
reg[rd1] := zero_extend(mem[addr+3:addr])
reg[rd2] := zero_extend(mem[addr+7:addr+3])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding `LWU` instructions would trigger.

Included in

Extension
XTheadMemPair (Chapter 10)

10.1.4. th.sdd

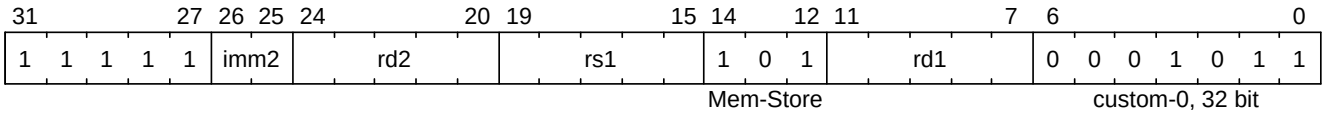
Synopsis

Store two 64-bit values to memory from two GPRs.

Mnemonic

th.sdd *rd1*, *rd2*, (*rs1*), *imm2*, 4

Encoding



Description

This instruction stores two 64-bit values from the two GP registers *rd1* and *rd2* to the address *rs1* + (zero_extend(*imm2*) << 4).

Operation

```
addr := rs1 + (zero_extend(imm2) << 4)
mem[addr+7:addr] := reg[rd1]
mem[addr+15:addr+8] := reg[rd2]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding SD instructions would trigger.

Included in

Extension
XTheadMemPair (Chapter 10)

10.1.5. th.swd

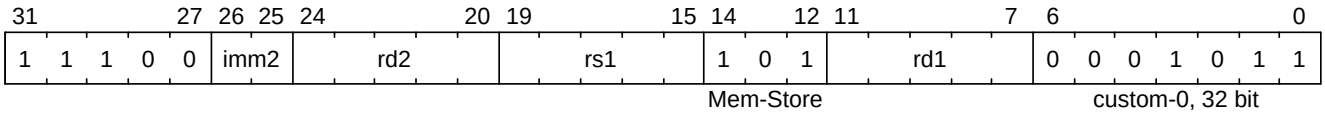
Synopsis

Store two 32-bit values to memory from two GPRs.

Mnemonic

th.swd *rd1*, *rd2*, (*rs1*), *imm2*, 3

Encoding



Description

This instruction loads two 32-bit values into the two GP registers *rd1* and *rd2* from the address *rs1* + (zero_extend(*imm2*) << 3).

Operation

```
addr := rs1 + (zero_extend(imm2) << 3)
mem[addr+3:addr] := reg[rd1][31:0]
mem[addr+7:addr+3] := reg[rd2][31:0]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding `SW` instructions would trigger.

Included in

Extension
XTheadMemPair (Chapter 10)

Chapter 11. Indexed memory operations for floating-point registers (XTheadFMemIdx)

The `XTheadFMemIdx` extension is `stable`.

The `XTheadFMemIdx` ISA extension provides indexed memory operations for floating-point registers.

The table below gives an overview of the instructions:

F	D	Mnemonic	Instruction
Y	Y	th.flrd *rd*, *rs1*, *rs2*, *imm2*	Load indexed double
Y	Y	th.flrw *rd*, *rs1*, *rs2*, *imm2*	Load indexed float
N	Y	th.flurd *rd*, *rs1*, *rs2*, *imm2*	Load unsigned indexed double
N	Y	th.flurw *rd*, *rs1*, *rs2*, *imm2*	Load unsigned indexed float
Y	Y	th.fsrd *rd*, *rs1*, *rs2*, *imm2*	Store indexed double
Y	Y	th.fsrw *rd*, *rs1*, *rs2*, *imm2*	Load indexed float
N	Y	th.fsurd *rd*, *rs1*, *rs2*, *imm2*	Store unsigned indexed double
N	Y	th.fsurw *rd*, *rs1*, *rs2*, *imm2*	Load unsigned indexed float

All instructions are available for `RV32` and `RV64`. Additionally at least the `F` extension needs to be available. In order to have all instructions available, the `D` extensions needs to be implemented.

11.1. Instructions

11.1.1. th.flrd

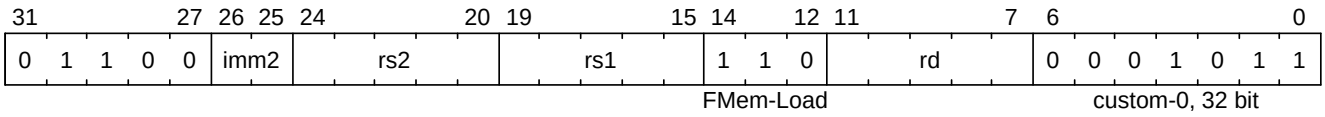
Synopsis

Load indexed double-precision floating point value.

Mnemonic

th.flrd *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a double-precision floating point value into FP register *rd* from the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
rd := fmem[addr+7:addr]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding `FLD` instructions would trigger.

Included in

Extension
XTheadFMemIdx (Chapter 11)

11.1.2. th.flrw

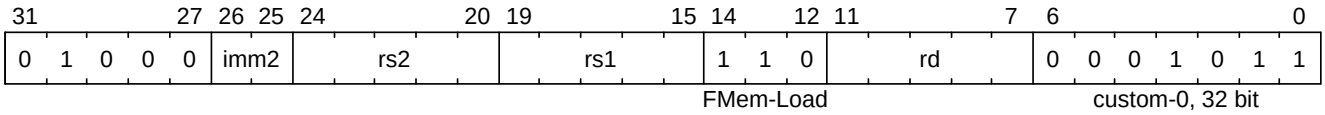
Synopsis

Load indexed single-precision floating point value.

Mnemonic

th.flrw *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a single-precision floating point value into FP register *rd* from the address *rs1* + (*rs2* << *imm2*).

If the `D` extension is available the value will be one-extended.

Operation

```
addr := rs1 + (rs2 << imm2)
rd := one_extend(fmem[addr+3:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding `FLW` instructions would trigger.

Included in

Extension
XTheadFMemIdx (Chapter 11)

11.1.3. th.flurd

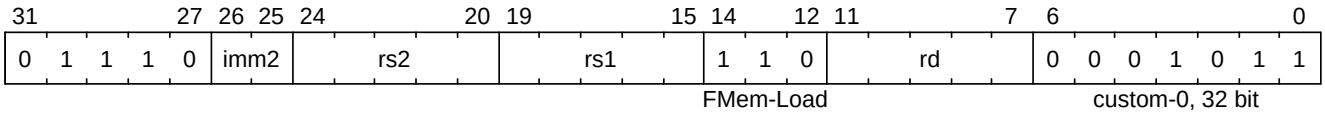
Synopsis

Load unsigned indexed double-precision floating point value.

Mnemonic

th.flurd *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a double-precision floating point value into FP register *rd* from the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.flrd` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
rd := fmem[addr+7:addr]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding `FLD` instructions would trigger.

Included in

Extension
XTheadFMemIdx (Chapter 11)

11.1.4. th.flurw

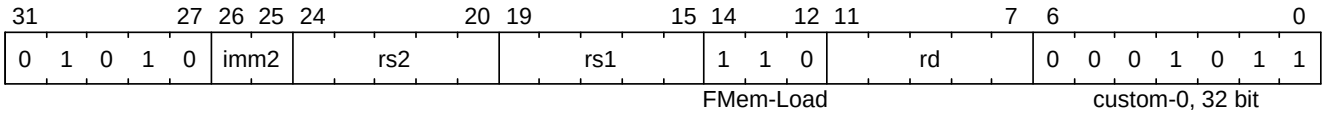
Synopsis

Load unsigned indexed single-precision floating point value.

Mnemonic

th.flurw *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction loads a single-precision floating point value into FP register *rd* from the address *rs1* + (zero_extend(*rs2*) << *imm2*).

If the `D` extension is available the value will be one-extended.

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.flrw` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
rd := one_extend(fmem[addr+3:addr])
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `FLW` instructions would trigger.

Included in

Extension
XTheadFMemIdx (Chapter 11)

11.1.5. th.fsrd

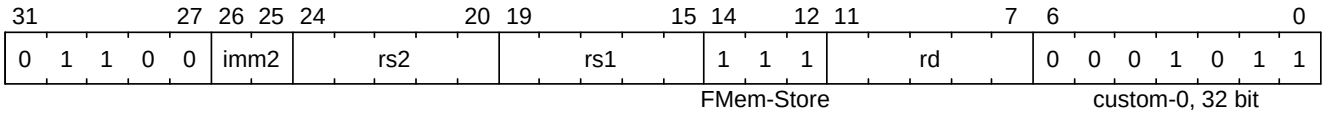
Synopsis

Store indexed double-precision floating point value.

Mnemonic

th.fsrd *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores a double-precision floating point value from the FP register *rd* to the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
fmem[addr+7:addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding `FSD` instructions would trigger.

Included in

Extension
XTheadFMemIdx (Chapter 11)

11.1.6. th.fsrw

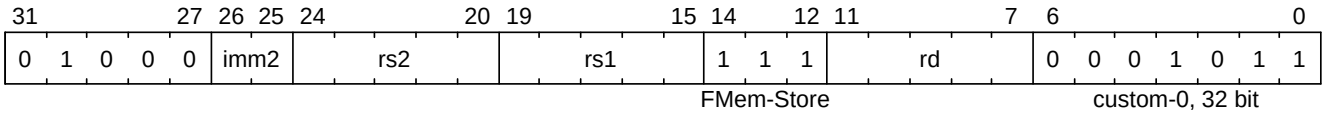
Synopsis

Store indexed single-precision floating point value.

Mnemonic

th.fsrw *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores a single-precision floating point value from the FP register *rd* to the address *rs1* + (*rs2* << *imm2*).

Operation

```
addr := rs1 + (rs2 << imm2)
fmem[addr+3:addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding `FSW` instructions would trigger.

Included in

Extension
XTheadFMemIdx (Chapter 11)

11.1.7. th.fsurd

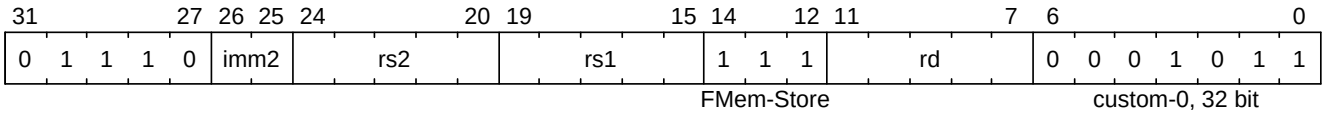
Synopsis

Store unsigned indexed double-precision floating point value.

Mnemonic

th.fsurd *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores a double-precision floating point value from FP register *rd* to the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.fsrd` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
fmem[addr+7:addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that two corresponding `FSD` instructions would trigger.

Included in

Extension
XTheadFMemIdx (Chapter 11)

11.1.8. th.fsurw

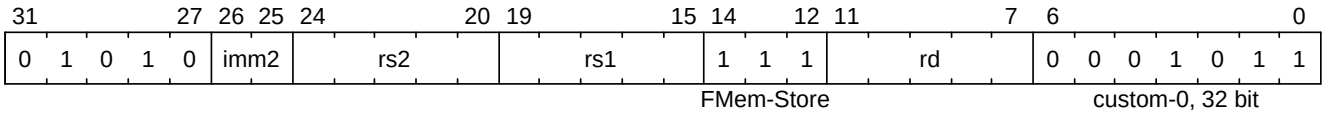
Synopsis

Store unsigned indexed single-precision floating point value.

Mnemonic

th.fsurw *rd*, *rs1*, *rs2*, *imm2*

Encoding



Description

This instruction stores a single-precision floating point value from FP register *rd* to the address *rs1* + (zero_extend(*rs2*) << *imm2*).

Note, that this instruction is equivalent to a `zext.w rs2, rs2` followed by a `th.fsrw` with the same arguments.

Operation

```
addr := rs1 + (zero_extend(rs2) << imm2)
fmem[addr+3:addr] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a corresponding `FLW` instructions would trigger.

Included in

Extension
XTheadFMemIdx (Chapter 11)

Chapter 12. Multiply-accumulate instructions (XTheadMac)

The `XTheadMac` extension is `stable`.

The `XTheadMac` ISA extension provides multiply-accumulate instructions.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
Y	Y	th.mula *rd*, *rs1*, *rs2*	Multiply-add double-words
Y	Y	th.mulah *rd*, *rs1*, *rs2*	Multiply-add half-words
N	Y	th.mulaw *rd*, *rs1*, *rs2*	Multiply-add words
Y	Y	th.muls *rd*, *rs1*, *rs2*	Multiply-subtract double-words
Y	Y	th.mulsh *rd*, *rs1*, *rs2*	Multiply-subtract half-words
Y	Y	th.mulsw *rd*, *rs1*, *rs2*	Multiply-subtract words

12.1. Instructions

12.1.1. th.mula

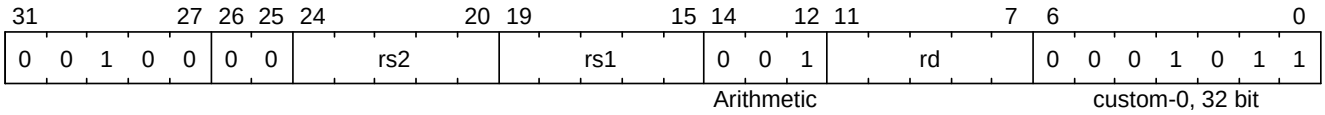
Synopsis

Compute multiply-add result of double-word operands.

Mnemonic

th.mula *rd*, *rs1*, *rs2*

Encoding



Description

This instruction computes the multiply-add result of the provided double-word operands.

Operation

```
M := reg[rs1] * reg[rs2]
reg[rd] := reg[rd] + M
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadMac (Chapter 12)

12.1.2. th.mulah

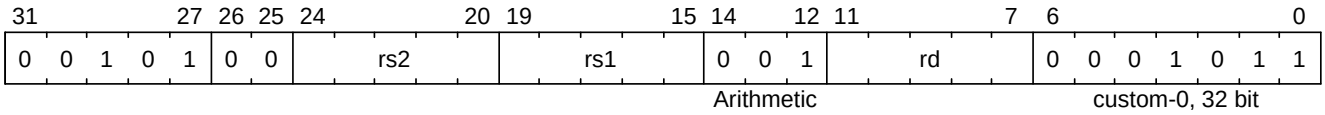
Synopsis

Compute multiply-add result of half-word operands.

Mnemonic

th.mulah *rd*, *rs1*, *rs2*

Encoding



Description

This instruction computes the multiply-add result of the provided half-word operands.

Operation

```
M := sext.w(reg[rs1][15:0]) * sext.w(reg[rs2][15:0])
reg[rd] := sext.w(reg[rd] + M)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadMac (Chapter 12)

12.1.3. th.mulaw

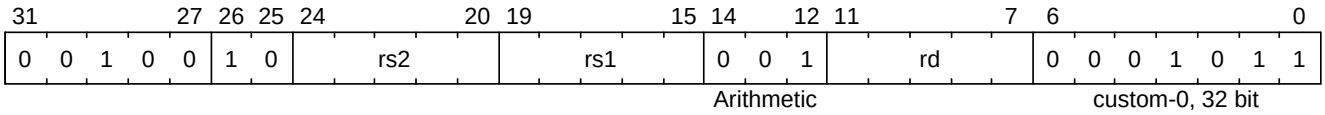
Synopsis

Compute multiply-add result of word operands.

Mnemonic

th.mulaw *rd*, *rs1*, *rs2*

Encoding



Description

This instruction computes the multiply-add result of the provided word operands.

Operation

```
M := sext.w(reg[rs1]) * sext.w(reg[rs2])
reg[rd] := sext.w(reg[rd] + M)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadMac (Chapter 12)

12.1.4. th.muls

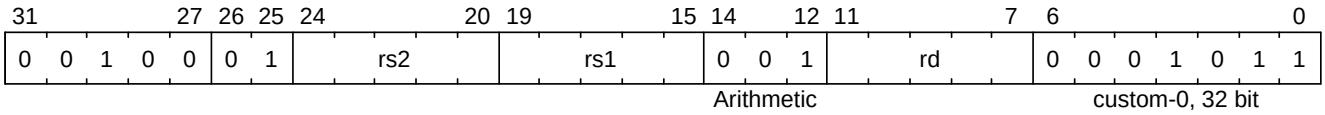
Synopsis

Compute multiply-subtract result of double-word operands.

Mnemonic

th.muls *rd*, *rs1*, *rs2*

Encoding



Description

This instruction computes the multiply-subtract result of the provided double-word operands.

Operation

```
M := reg[rs1] * reg[rs2]
reg[rd] := reg[rd] - M
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadMac (Chapter 12)

12.1.5. th.mulsh

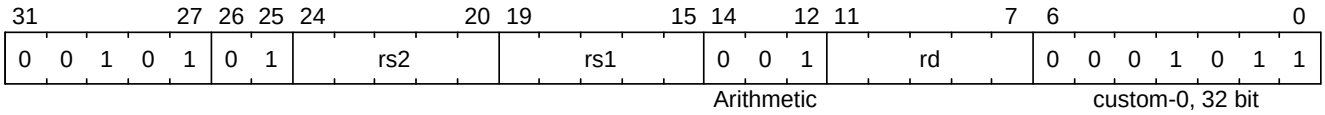
Synopsis

Compute multiply-subtract result of half-word operands.

Mnemonic

th.mulsh *rd*, *rs1*, *rs2*

Encoding



Description

This instruction computes the multiply-subtract result of the provided half-word operands.

Operation

```
M := sext.h(reg[rs1][15:0]) * sext.h(reg[rs2][15:0])
reg[rd] := sext.w(reg[rd] - M)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadMac (Chapter 12)

12.1.6. th.mulsw

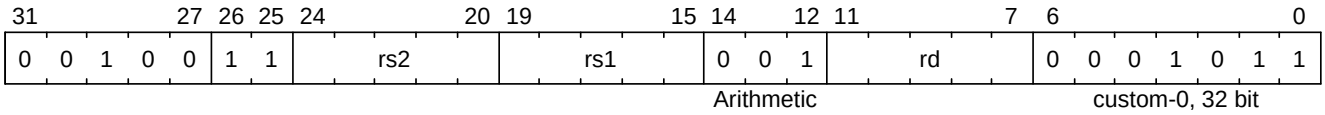
Synopsis

Compute multiply-subtract result of word operands.

Mnemonic

th.mulsw *rd*, *rs1*, *rs2*

Encoding



Description

This instruction computes the multiply-subtract result of the provided word operands.

Operation

```
M := sext.w(reg[rs1]) * sext.w(reg[rs2])
reg[rd] := sext.w(reg[rd] - M)
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadMac (Chapter 12)

Chapter 13. Double-precision floating-point high-bit data transmission instructions

The `XTheadFmv` extension is `stable`.

The `XTheadFmv` ISA extension provides double-precision floating-point high-bit data transmission intructions for RV32.

This extension depends on the availability of the `D` (double-precision floating-point) ISA extension.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
Y	N	th.fmv.hw.x *rd, fs1*	Write double-precision floating-point high-bit data
Y	N	th.fmv.x.hw *rd, fs1*	Read double-precision floating-point high-bit data

13.1. Instructions

13.1.1. th.fmv.x.hw

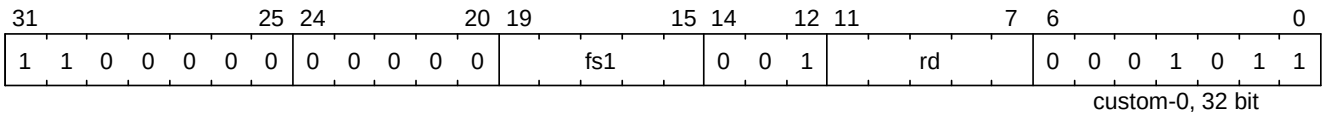
Synopsis

Read double-precision floating-point high-bit data

Mnemonic

th.fmv.x.hw *rd, fs1*

Encoding



Description

This instruction stores the upper 32-bit of the specified double-precision FP register *fs1* in the specified 32-bit GP register *rd*.

Operation

```
rd := fs1[63:32]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadFmv (Chapter 13)

13.1.2. th.fmv.hw.x

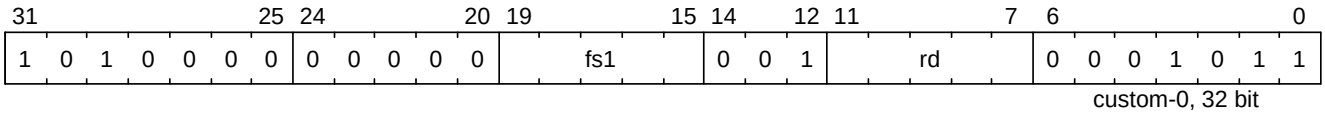
Synopsis

Write double-precision floating-point high-bit data

Mnemonic

th.fmv.hw.x *rd*, *fs1*

Encoding



Description

This instruction stores the contents of the specified 32-bit GP register *rd* in the upper 32-bit of the specified double-precision FP register *fs1*.

Operation

```
fs1[63:32] := rd
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadFmv (Chapter 13)

Chapter 14. Acceleration interruption instructions

The `XTheadInt` extension is `stable`.

The `XTheadInt` ISA extension provides instructions to reduce the code size of ISRs and/or the interrupt latencies that are caused by ISR entry/exit code.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
Y	Y	th.ipush	Push register context on interrupt stack
Y	Y	th.ipop	Pop register context from interrupt stack

14.1. Instructions

14.1.1. th.ipush

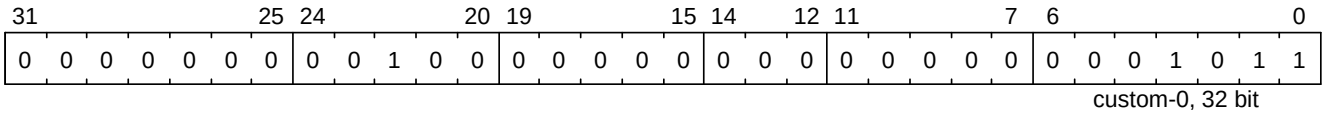
Synopsis

Pushes register context on the interrupt stack

Mnemonic

ipush

Encoding



Description

This instruction stores the following information on the interrupt stack:

- mcause
- mepc
- callee-saved registers (X1, X5-X7, X10-X17, and X28-X31)

Additionally the interrupt stack pointer is advanced accordingly.

Operation

```
mem[int_sp-4:int_sp-72] := {mcause, mepc, X1, X5-X7, X10-X17, X28-X31}
int_sp := int_sp - 72
```

Permission

This instruction can be executed in `M` mode only. Attempts to execute this instruction in other modes will raise an illegal instruction exception.

Exceptions

This isntruction may trigger an unaligned access exception or an access error exception.

Included in

Extension
XTheadFmv (Chapter 14)

14.1.2. th.ipop

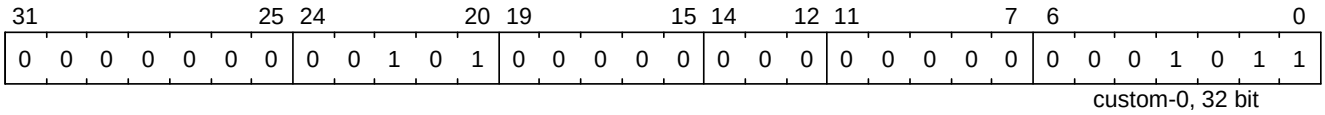
Synopsis

Pop register context from the interrupt stack

Mnemonic

ipop

Encoding



Description

This instruction restores the following information from the interrupt stack:

- mcause
- mepc
- callee-saved registers (X1, X5-X7, X10-X17, and X28-X31)

Additionally the interrupt stack pointer is moved back accordingly.

Operation

```
{mcause, mepc, X1, X5-X7, X10-X17, X28-X31} := mem[int_sp+68:int_sp]
int_sp := int_sp + 72
mret
```

Permission

This instruction can be executed in `M` mode only. Attempts to execute this instruction in other modes will raise an illegal instruction exception.

Exceptions

This instruction does not trigger any exceptions.

Included in

Extension
XTheadFmv (Chapter 14)

Chapter 15. Vector four 8-bit multiply and add with 32-bit instructions

The XTheadVdot extension is stable.

The XTheadVdot ISA extension provides vector integer four 8-bit multiply and add with 32-bit element intructions.

This extension depends on the availability of the V (vector) ISA extension.

The table below gives an overview of the instructions:

RV32	RV64	Mnemonic	Instruction
Y	Y	th.vmaqa.vv *vd, vs1, vs2*	Four signed 8-bit multiply with 32-bit add(vector-vector)
Y	Y	th.vmaqa.vx *vd, rs1, vs2*	Four signed 8-bit multiply with 32-bit add(vector-scalar)
Y	Y	th.vmaqau.vv *vd, vs1, vs2*	Four unsigned 8-bit multiply with 32-bit add(vector-vector)
Y	Y	th.vmaqau.vx *vd, rs1, vs2*	Four unsigned 8-bit multiply with 32-bit add(vector-scalar)
Y	Y	th.vmaqasu.vv *vd, vs1, vs2*	Four signed-unsigned and 8-bit multiply with 32-bit add(vector-vector)
Y	Y	th.vmaqasu.vx *vd, rs1, vs2*	Four signed-unsigned and 8-bit multiply with 32-bit add(vector-scalar)
Y	Y	th.vmaqaus.vx *vd, rs1, vs2*	Four unsigned-signed and 8-bit multiply with 32-bit add(vector-scalar)

15.1. Instructions

15.1.1. th.vmaqa.vv

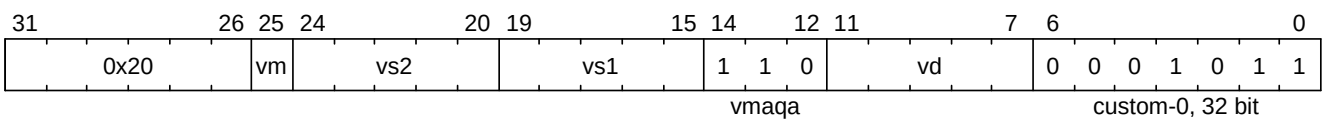
Synopsis

Four signed 8-bit multiply with 32-bit add.

Mnemonic

th.vmaqa.vv *vd, vs1, vs2*

Encoding



Description

The four signed 8-bit elements of 32-bit of vs1 are multiplied with the four signed 8-bit elements of 32-bit of vs2 and then the four results are added together with the corresponding 32-bit element of Vd. This instruction is based on vector extension.The vector masking operates at source operands with 8-bit element size. If vm=1, the instruction is unmasked and the instruction is vmaqa.vv vd, vs1, vs2. If vm=0, the instruction is masked and the instruction is vmaqa.vv vd, vs1, vs2, v0.t. When v0.mask[i] is 1, the multiplication result of vs1[(i+1)*8:i*8] and vs2[(i+1)*8:i*8] is added with vd.The vector length(vl) operates at destination operands with 32-bit element size.

Operation

```
vd[i] = vd[i] + vs1[i][7:0]   * vs2[i][7:0]
              + vs1[i][15:8]  * vs2[i][15:8]
              + vs1[i][23:16] * vs2[i][23:16]
              + vs1[i][31:24] * vs2[i][31:24]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a `vmacc.vv` instructions would trigger except that the value of vsew[2:0] must be 3'b010.

Included in

Extension
XTheadvdot (Chapter 15)

15.1.2. th.vmaqa.vx

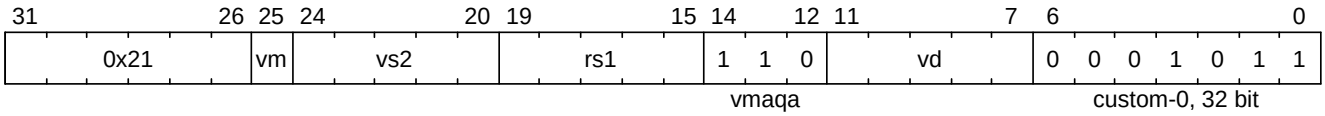
Synopsis

Four signed 8-bit multiply with 32-bit add.

Mnemonic

th.vmaqa.vx *vd*, *rs1*, *vs2*

Encoding



Description

The four signed 8-bit elements of the lower 32-bit of rs1 are multiplied with the four signed 8-bit elements of each 32-bit of vs2 and then the four results are added together with the corresponding 32-bit element of Vd. This instruction is based on vector extension.The vector masking operates at source operands with 8-bit element size. If vm=1, the instruction is unmasked and the instruction is vmaqa.vx vd, rs1, vs2. If vm=0, the instruction is masked and the instruction is vmaqa.vx vd, rs1, vs2, v0.t. When v0.mask[i] is 1, the multiplication result of rs1[(i+1)*8:i*8] and vs2[(i+1)*8:i*8] is added with vd.The vector length(vl) operates at destination operands with 32-bit element size.

Operation

```
vd[i] = vd[i] + rs1[7:0]   * vs2[i][7:0]
              + rs1[15:8]  * vs2[i][15:8]
              + rs1[23:16] * vs2[i][23:16]
              + rs1[31:24] * vs2[i][31:24]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a `vmacc.vv` instructions would trigger except that the value of vsew[2:0] must be 3'b010.

Included in

Extension
XTheadvdot (Chapter 15)

15.1.3. th.vmaqau.vv

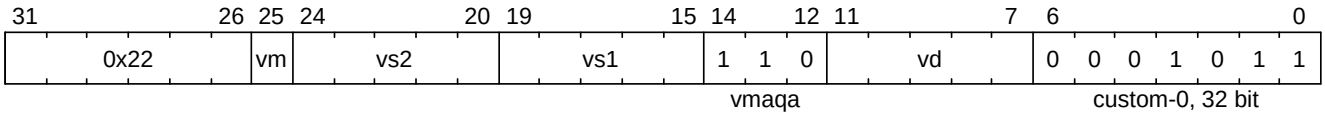
Synopsis

Four unsigned 8-bit multiply with 32-bit add.

Mnemonic

th.vmaqau.vv *vd*, *vs1*, *vs2*

Encoding



Description

The four unsigned 8-bit elements of 32-bit of vs1 are multiplied with the four unsigned 8-bit elements of 32-bit of vs2 and then the four results are added together with the corresponding 32-bit element of Vd. This instruction is based on vector extension.The vector masking operates at source operands with 8-bit element size. If vm=1, the instruction is unmasked and the instruction is vmaqau.vv vd, vs1, vs2. If vm=0, the instruction is masked and the instruction is vmaqau.vv vd, vs1, vs2, v0.t. When v0.mask[i] is 1, the multiplication result of vs1[(i+1)*8:i*8] and vs2[(i+1)*8:i*8] is added with vd.The vector length(vl) operates at destination operands with 32-bit element size.

Operation

```
vd[i] = vd[i] + vs1[i][7:0]   * vs2[i][7:0]
              + vs1[i][15:8]  * vs2[i][15:8]
              + vs1[i][23:16] * vs2[i][23:16]
              + vs1[i][31:24] * vs2[i][31:24]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a `vmacc.vv` instructions would trigger except that the value of vsew[2:0] must be 3'b010.

Included in

Extension
XTheadvdot (Chapter 15)

15.1.4. th.vmaqau.vx

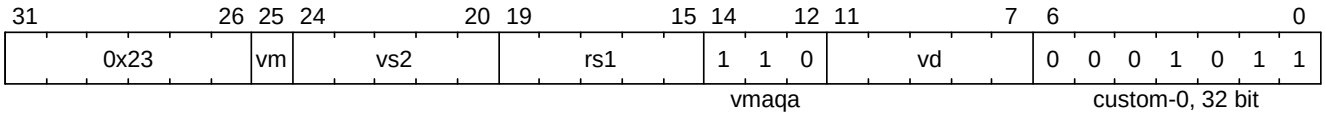
Synopsis

Four unsigned 8-bit multiply with 32-bit add.

Mnemonic

th.vmaqau.vx *vd*, *rs1*, *vs2*

Encoding



Description

The four unsigned 8-bit elements of the lower 32-bit of rs1 are multiplied with the four unsigned 8-bit elements of each 32-bit of vs2 and then the four results are added together with the corresponding 32-bit element of Vd. This instruction is based on vector extension.The vector masking operates at source operands with 8-bit element size. If vm=1, the instruction is unmasked and the instruction is vmaqau.vx vd, rs1, vs2. If vm=0, the instruction is masked and the instruction is vmaqau.vx vd, rs1, vs2, v0.t. When v0.mask[i] is 1, the multiplication result of rs1[(i+1)*8:i*8] and vs2[(i+1)*8:i*8] is added with vd.The vector length(vl) operates at destination operands with 32-bit element size.

Operation

```
vd[i] = vd[i] + rs1[7:0]   * vs2[i][7:0]
              + rs1[15:8]  * vs2[i][15:8]
              + rs1[23:16] * vs2[i][23:16]
              + rs1[31:24] * vs2[i][31:24]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a `vmacc.vv` instructions would trigger except that the value of vsew[2:0] must be 3'b010.

Included in

Extension
XTheadvdot (Chapter 15)

15.1.5. th.vmaqasu.vv

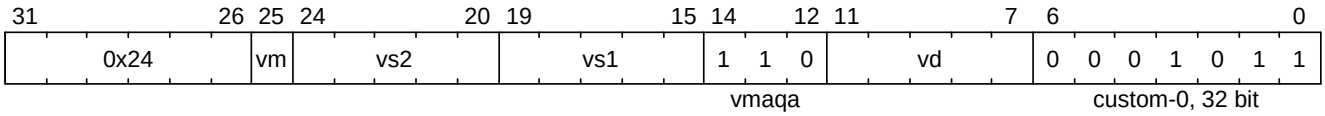
Synopsis

Four signed-unsigned 8-bit multiply with 32-bit add.

Mnemonic

th.vmaqasu.vv *vd*, *vs1*, *vs2*

Encoding



Description

The four signed 8-bit elements of 32-bit of vs1 are multiplied with the four unsigned 8-bit elements of 32-bit of vs2 and then the four results are added together with the corresponding 32-bit element of Vd. This instruction is based on vector extension.The vector masking operates at source operands with 8-bit element size. If vm=1, the instruction is unmasked and the instruction is vmaqasu.vv vd, vs1, vs2. If vm=0, the instruction is masked and the instruction is vmaqasu.vv vd, vs1, vs2, v0.t. When v0.mask[i] is 1, the multiplication result of vs1[(i+1)*8:i*8] and vs2[(i+1)*8:i*8] is added with vd.The vector length(vl) operates at destination operands with 32-bit element size.

Operation

```
vd[i] = vd[i] + vs1[i][7:0]   * vs2[i][7:0]
              + vs1[i][15:8]  * vs2[i][15:8]
              + vs1[i][23:16] * vs2[i][23:16]
              + vs1[i][31:24] * vs2[i][31:24]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a `vmacc.vv` instructions would trigger except that the value of vsew[2:0] must be 3'b010.

Included in

Extension
XTheadvdot (Chapter 15)

15.1.6. th.vmaqasu.vx

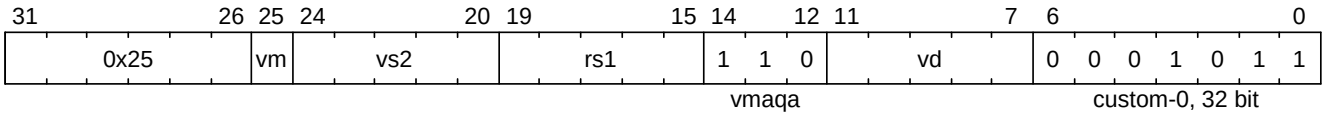
Synopsis

Four signed-unsigned 8-bit multiply with 32-bit add.

Mnemonic

th.vmaqasu.vx *vd*, *rs1*, *vs2*

Encoding



Description

The four signed 8-bit elements of the lower 32-bit of rs1 are multiplied with the four unsigned 8-bit elements of each 32-bit of vs2 and then the four results are added together with the corresponding 32-bit element of Vd. This instruction is based on vector extension.The vector masking operates at source operands with 8-bit element size. If vm=1, the instruction is unmasked and the instruction is vmaqasu.vx vd, rs1, vs2. If vm=0, the instruction is masked and the instruction is vmaqasu.vx vd, rs1, vs2, v0.t. When v0.mask[i] is 1, the multiplication result of rs1[(i+1)*8:i*8] and vs2[(i+1)*8:i*8] is added with vd.The vector length(vl) operates at destination operands with 32-bit element size.

Operation

```
vd[i] = vd[i] + rs1[7:0]   * vs2[i][7:0]
              + rs1[15:8]  * vs2[i][15:8]
              + rs1[23:16] * vs2[i][23:16]
              + rs1[31:24] * vs2[i][31:24]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a `vmacc.vv` instructions would trigger except that the value of vsew[2:0] must be 3'b010.

Included in

Extension
XTheadvdot (Chapter 15)

15.1.7. th.vmaqaus.vx

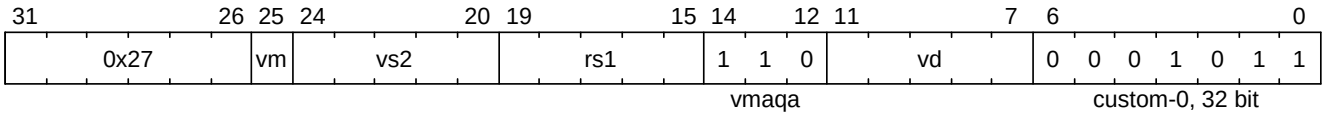
Synopsis

Four unsigned-signed 8-bit multiply with 32-bit add.

Mnemonic

th.vmaqaus.vx *vd*, *rs1*, *vs2*

Encoding



Description

The four unsigned 8-bit elements of the lower 32-bit of rs1 are multiplied with the four signed 8-bit elements of each 32-bit of vs2 and then the four results are added together with the corresponding 32-bit element of Vd. This instruction is based on vector extension.The vector masking operates at source operands with 8-bit element size. If vm=1, the instruction is unmasked and the instruction is vmaqaus.vx vd, rs1, vs2. If vm=0, the instruction is masked and the instruction is vmaqaus.vx vd, rs1, vs2, v0.t. When v0.mask[i] is 1, the multiplication result of rs1[(i+1)*8:i*8] and vs2[(i+1)*8:i*8] is added with vd.The vector length(vl) operates at destination operands with 32-bit element size.

Operation

```
vd[i] = vd[i] + rs1[7:0]   * vs2[i][7:0]
              + rs1[15:8]  * vs2[i][15:8]
              + rs1[23:16] * vs2[i][23:16]
              + rs1[31:24] * vs2[i][31:24]
```

Permission

This instruction can be executed in all privilege levels.

Exceptions

This instruction triggers the same exceptions that a `vmacc.vv` instructions would trigger except that the value of vsew[2:0] must be 3'b010.

Included in

Extension
XTheadvdot (Chapter 15)